ISSUE 23 - MAY 2014

Get printed copies at themagpi.com

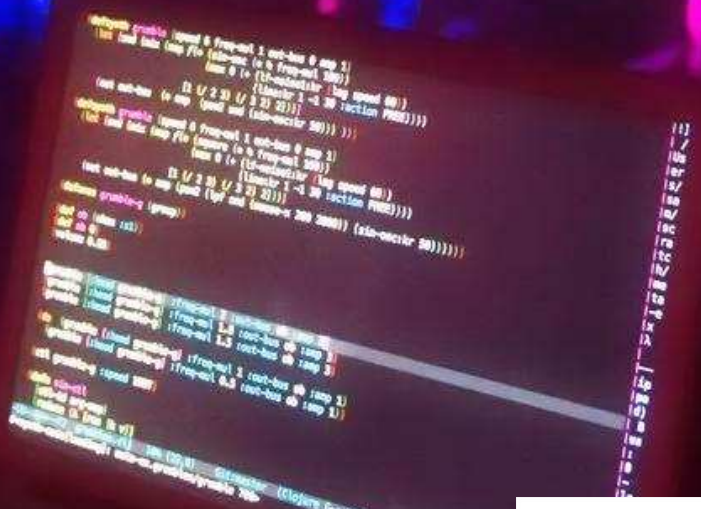
The MagPi

A Magazine for Raspberry Pi Users

SONIC PI 2.0
Get Your Groove On

High Altitude Measuring
Real World Minecraft
BrickPi and Scratch
XMPP Chat Server
Linux Tool Shed
1-Wire Sensors
GrovePi

BIG BIRTHDAY COMPETITION
Win more than **£2,000** of goodies!



ISSN 2051-9990

Raspberry Pi is a trademark of The Raspberry Pi Foundation.
This magazine was created using a Raspberry Pi computer.

CC BY NC SA

The MagPi

http://www.themagpi.com

The MagPi 23 TM

Welcome to Issue 23 of The MagPi magazine.

It's party time here at The MagPi towers, celebrating our second birthday! To mark this milestone, The MagPi is pleased to provide another massive chance to get your hands on some fantastic Raspberry Pi goodies, with over £2000 worth of tasty treats for our readers to win! Thank you to all our sponsors who have kindly given towards this massive collection of prizes. See pages 18-19 for more information.

This month you'll find us in the club with Sonic Pi. The MagPi has an exclusive of Samuel Aaron's brand new release of Sonic Pi v2.0 and how it is aiding build the underground music movement of Live Coding. Samuel describes what is new to v2.0 along with some basics to get you up and mixing in no time.

Jacob Marsh from ModMyPi is back with another great tutorial on physical computing, this month describing how to use 1-Wire temperature sensors. We look at how to build your own XMPP chat server in Gianluca's Chat Room article, then Bernhard Suter provides the next article in our Linux tool shed series where he describes the bash shell.

Michael Petersen begins his two-part series on using the Raspberry Pi to study atmospheric pollution. In this article he introduces us to the main subsystems involved in the multi-sensor array used in the research balloons which are sent into the lower stratosphere of Utah. We also take a look at stackable hardware with Sai Yamanoor's article about GrovePi.

Our very own William Bell has been working overtime this month with no less than three articles. First, he shows how to interface BrickPi with Scatch, then he describes how to bring Minecraft to the real world and finally the C++ Cache series makes a welcome return with an explanation of classes.

Why not head over to our Facebook page http://www.facebook.com/MagPiMagazine and let us know your favourite article over the last two years, or even what you want to read about over the next 12 months!

Enjoy.

Chief Editor of The MagPi

The MagPi Team

Ash Stone - Chief Editor / Administration / Layout
W.H. Bell - Issue Editor / Layout / Administration
Bryan Butler - Page Design / Graphics
Ian McAlpine - Layout / Testing / Proof Reading
Matt Judge - Website / Administration
Aaron Shaw - Layout
Nick Hitch - Admin
Colin Deady - Layout / Testing / Proof Reading
Tim Cox - Testing / Proof Reading
Claire Price - Layout / Proof Reading
Chris Stag - Testing

Contents

4 **STUDYING ATMOSPHERIC POLLUTION WITH A MULTI-SENSOR ARRAY**
Part 1: Introduction to the main subsystems

8 **GROVEPI: ADDING GROVE SENSOR MODULES**
Stackable hardware extension board

12 **BRICKPI**
Part 3: Scratch interface with RpiScratchIO

18 **BIG BIRTHDAY COMPETITION**
Over £2000 of prizes to be won in our second anniversary competition

20 **MINECRAFT PI EDITION**
Part 2: Interfacing Minecraft with PiFace Digital

26 **PHYSICAL COMPUTING**
Part 2: Using 1-Wire temperature sensors

30 **C++ CACHE**
Part 5: Classes

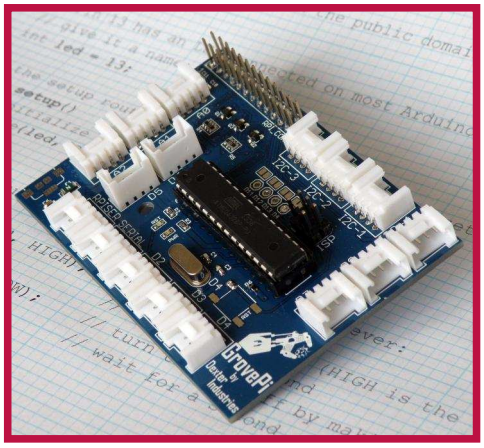
34 **CHAT ROOM**
Turn your Raspberry Pi into an XMPP chat server

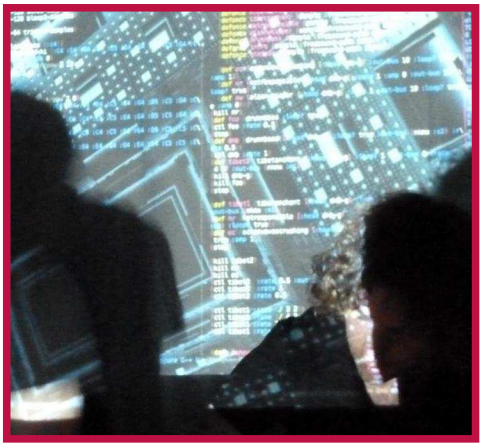
38 **LINUX COMMANDS**
Part 2: Tales from the Linux tool shed - don't bash the shell

42 **THIS MONTH'S EVENTS GUIDE**
Cambridge UK, North Staffordshire UK, Cardiff UK, San Mateo CA USA, Bath UK

44 **SONICPI: GET YOUR GROOVE ON!**
Part 2: Discover new samples, synths, studio effects and Live Coding

48 **FEEDBACK**
Have your say about The MagPi





STUDYING ATMOSPHERIC POLLUTION

A Multi-Sensor Array for Atmospheric Science

Michael Petersen

Guest Writer

Part 1: Introduction to the multi-sensor array subsystems

SKILL LEVEL : ADVANCED

This two part series describes the design and construction of a Multi-Sensor Array (MSA) for studying atmospheric pollution in an urbanised mountain basin; specifically, the region above Salt Lake City, Utah (USA). The MSA is flown on research balloons by HARBOR, an undergraduate research group located at Weber State University in Ogden, Utah. The MSA produces a column of measurements from ground level (approx. 1km above sea level, ASL) to the lower stratosphere (approx. 35km ASL).

During flight, the system is exposed to pressures as low as 0.75 mmHg, where heat exchange becomes difficult even at temperatures of -50°C. Jet stream winds can exceed speeds of 200km/h, applying punishing shock and vibration forces to our electronic equipment. In this extreme environment, the MSA must continue to gather scientific data for a minimum of 4 hours.

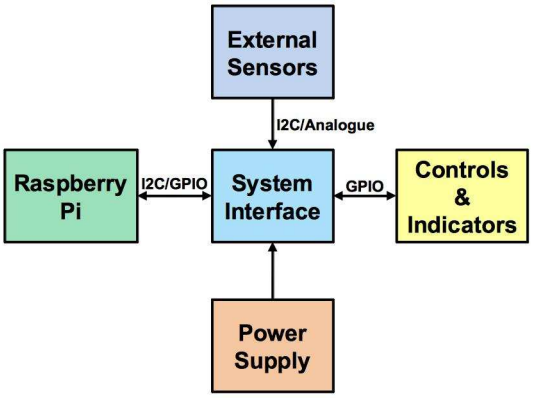
The first part of this series focuses on the hardware design of the MSA system. Part 2 will describe how the software was designed and implemented.

Theory of operation

The MSA is a data logging computer with a standard set of built-in sensors and an expanded set of inputs to allow guest packages to be connected. External devices such as LEDs, buzzers, pumps, and heaters are controlled via existing GPIO pins. The MSA is comprised of six major subsystems:

1. Raspberry Pi (Version 1 - Model B)
2. System Interface Daughterboard
3. External Sensor Board
4. External Controls and Indicators
5. Power Supply
6. Enclosures

The Raspberry Pi serves as the primary platform for the MSA system. It monitors inputs, communicates with sensors, processes and stores flight data, and controls LEDs, motors, and heaters.

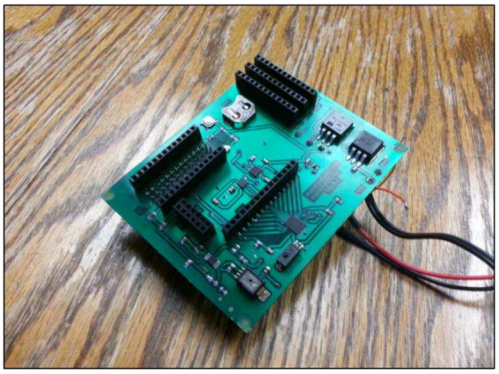


System Interface Daugherboard

The System Interface Daughterboard (SID) links the Raspberry Pi to everything else in the system. It also provides regulated power. The SID is where analogue and digital sensors are mounted or connected. It is equipped with the following components:

* MPU6000 (6-axis accelerometer/gyroscope)
* HMC5883L (3-axis magnetometer)
* MPX2102A (pressure sensor)
* MAX4208 (instrument amplifier)
* HIH5030 (humidity sensor)
* TMP112 (temperature sensor)
* COM-08720 (pushbutton)
* MIC2937A (3.3V regulator)
* MIC2937A (5.0V regulator)
* LTC2495 (16 Channel ADC)
* DS3231M (Real-Time Clock)

The sensors are used to calibrate data and help eliminate bad data which may have been recorded during an event; such as an impact, extreme temperature, or high humidity. All sensors communicate over the available I2C bus, with analogue sensors routed to an I2C capable ADC.



There are two primary sensor circuits built into the SID. The first is a flight dynamic circuit which consists of an I2C accelerometer/gyroscope sensor and magnetometer.

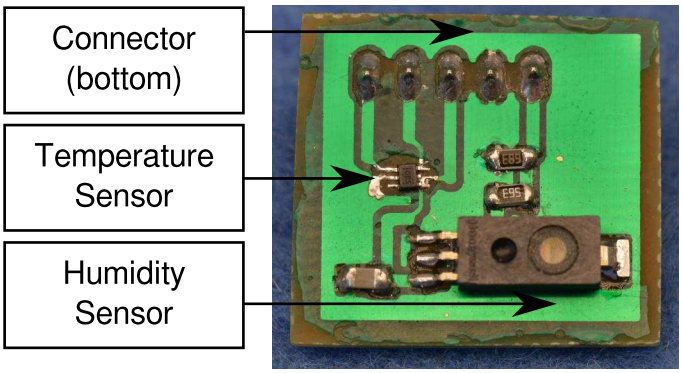
The second is an environmental circuit composed of temperature, humidity, and pressure sensors.

The ADC can support up to sixteen single-ended analogue sensors, not including an internal temperature sensor. The I2C bus can support many more devices.

We designed our circuit boards using a free version of the Eagle PCB Design Software by CadSoft. We manufactured the boards in-house using commercially available chemicals and photosensitive PCBs. The top view of a completed SID board is shown above. The assembled board mounts to the Raspberry Pi via the 2x13 pin header.

External sensors

The external sensor board consists of a TMP112 temperature sensor and an HIH5030 humidity sensor mounted to a small PCB embedded in the exterior wall of the external enclosure and connected with a short harness.



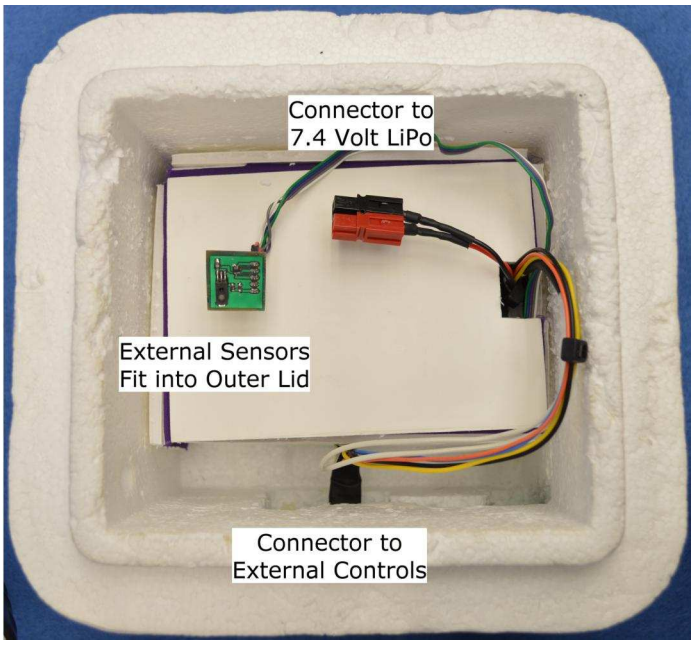
External controls and indicators

External controls and indicators allow the user to power up and shutdown the MSA from outside its insulated enclosure. LEDs and buzzers allow the user to know the status of the MSA, whether it is in standby or data logging mode, and also alerts unaware people on the ground when it is descending near the surface. A pull-pin starts a mission timer to allow synchronisation of data between various instruments at the start of a flight. It can also be re-inserted to save data files and initiate a safe-shutdown.

Power supply

The power supply subsystem combines software and hardware to provide power and initiate a safe shutdown in the event of battery failure.

The hardware component is basically a 7.4V LiPO battery and two linear voltage regulators (5V and 3.3V). The 5V regulator supplies power to the Raspberry Pi and has a couple hundred milliamps available for other devices. The 3.3V regulator provides voltage for the I2C bus and all of the onboard sensors.



Battery power is a precious commodity on balloon flights, as capacity per gram quickly eats into mass budgets. Federal regulations limit balloon packages to 5.4kg, which must be distributed amongst various flight packages; including radio transmitters, cameras, and other scientific instruments. The MSA system was limited to just 408g which made battery selection crucial.

The Raspberry Pi draws between 310mA and 390mA under normal operation; this includes the current draw of the HDMI output chip and the USB/Ethernet chips, which account for nearly one half of the total current consumption. To prolong battery life during flight, these chips must be disabled. The chips are disabled in the MSA code by calling two Linux commands.

The C code to shut down the HDMI video chip is:

```
system("/opt/vc/bin/tvservice -off");
```

The next line of code suspends the USB/Ethernet controller:

```
system("sh -c \"echo 1 >
/sys/devices/platform/bcm2708/usb/bussuspe
nd\" ");
```

Both of these chips are re-enabled when the MSA is rebooted or powered up for the first time. After the chips are disabled, the average current draw of the Raspberry Pi drops to 180mA.

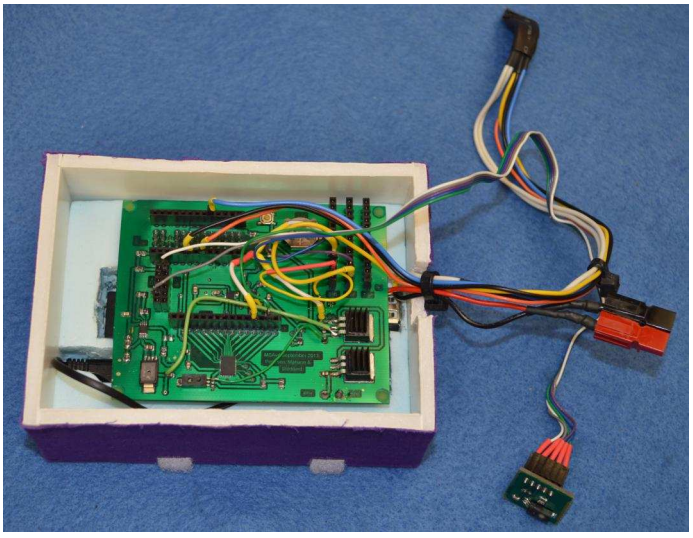
Factoring in the additional power consumption of the sensors and ICs, a total of 14.42mA, and the required battery life of 4 hours, a minimum battery capacity of 778mAh is required.

4hrs x (180mA + 14.42 mA) = 778mAh

We selected a 7.4V 1100mAh LiPO battery, which provides a modest amount of headroom and only costs us 72g of our 408g mass budget.

Enclosures

An internal enclosure was designed to protect the Raspberry Pi and SID from shock and vibrations as the MSA makes its way through the jet stream and upon landing. It is made of high-density foam wrapped in a lightweight fabric. The fabric acts as a hinge and allows the lid to wrap around the sides, which are secured with VELCRO® brand hook-and-loop fasterners.



The internal enclosure fits snuggly into a 30cm x 30cm x 30cm Styrofoam external enclosure. The external enclosure provides structural rigidness for the external controls, and insulates sensitive components from extreme environmental conditions.

Completely assembled, the whole unit weighs 406.9 grams. It has been successfully tested on four flights and has reached an altitude of 34 km. It has survived landing in rugged desert terrain and also a splashdown in a reservoir.

Coming up

In part 2 of this series, I will discuss how the MSA initiates the data logging program at startup, communicates with some of the sensors, stores data, manages sampling rates, interfaces with the user, and responds to conditional events; such as low battery voltage.

The code for the MSA is written completely in C and is facilitated with the aid of the BCM 2835 C library, maintained by Mike McCauley at http://www.airspayce.com. The code also takes advantage of semaphores and parallel programming to create a timer without the use of interrupts.

Acknowledgements

The following members contributed to the design and construction of the MSA.

Electronics Engineering Students:
- Michael Petersen
- J. Wesley Mahurin
- Jenifer Stoddard

Dr. John E. Sohl: Director of HARBOR and Professor of Physics at WSU
Dr. Fon Brown: Mentor and Professor of Electronics Engineering at WSU

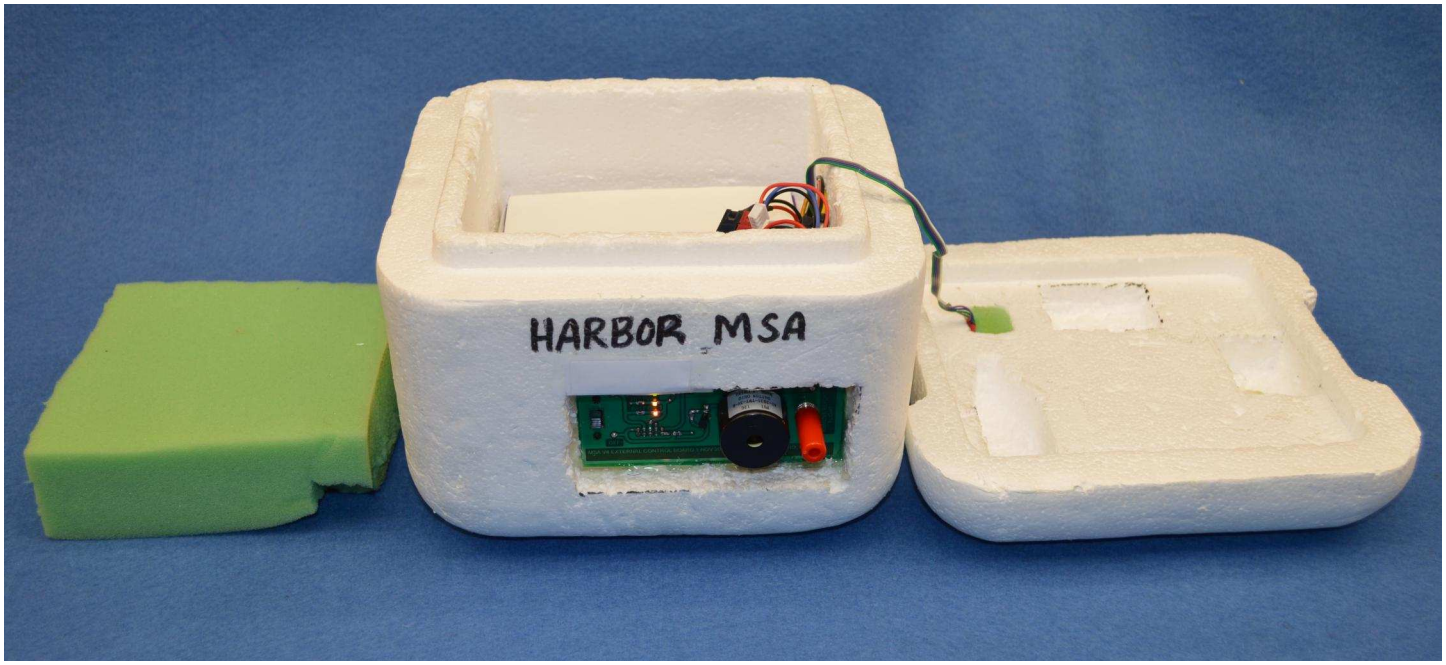
Development of the MSA was funded in part by the Val A. Browning Foundation, the Ralph Nye Charitable Foundation, the Weber State University Office of Undergraduate Research, and the Utah Chapter of the AIAA. HARBOR would like to give special thanks to Al Rydman for his hospitality and for making the Duchesne Municipal Airport available to us to launch our system.

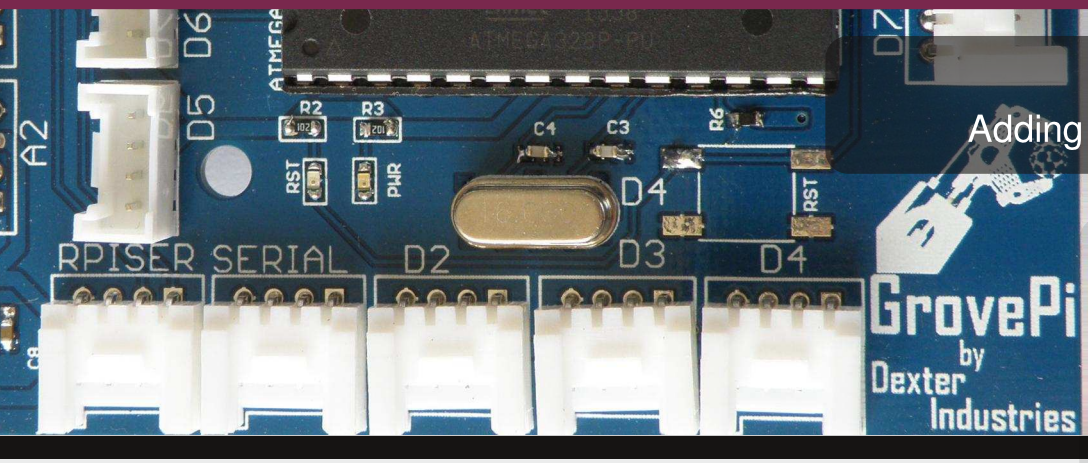
Useful links

CadSoft EAGLE PCB Design Software:
http://www.cadsoftusa.com/eagle-pcb-design-software

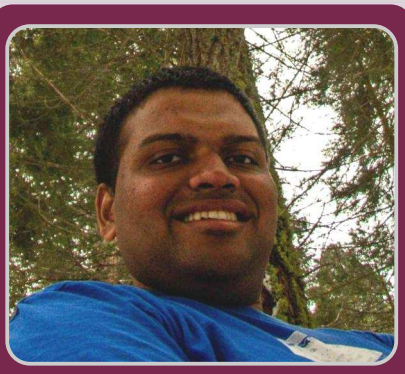
BCM 2835 C library:
http://airspayce.com/mikem/bcm2835/

HARBOR home page:
http://planet.weber.edu/harbor/





Stackable hardware extension board



Sai Yamanoor

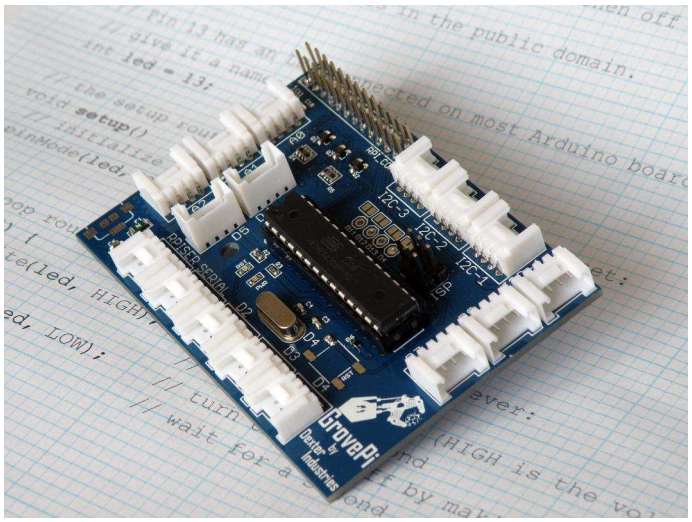
MagPi writer

SKILL LEVEL : INTERMEDIATE

Introduction

The GrovePi is a an extension board that allows additional hardware to be stacked on top of the Raspberry Pi.
http://www.dexterindustries.com/GrovePi/

The board simplifies the art of electronic circuit hacking, since it comes with an onboard microcontroller. The extension board enables interfacing the Raspberry Pi to an ecosystem of modules including sensors, actuators and displays, through a set of standard connectors. The GrovePi comes with analogue inputs, digital input/output (I/O) and I²C interfaces. The modules are called “electronic bricks”. The board is based on an Atmega328 microcontroller that comes preloaded with the firmware required to communicate with the Raspberry Pi. It also possible to flash the onboard microcontroller with your own firmware for a project. The GrovePi communicates with the Raspberry Pi using the I²C interface. Since the Grove Pi is stackable, it is possible to make use of other GPIO pins, as well as other devices that are connected to the Raspberry Pi I²C bus.

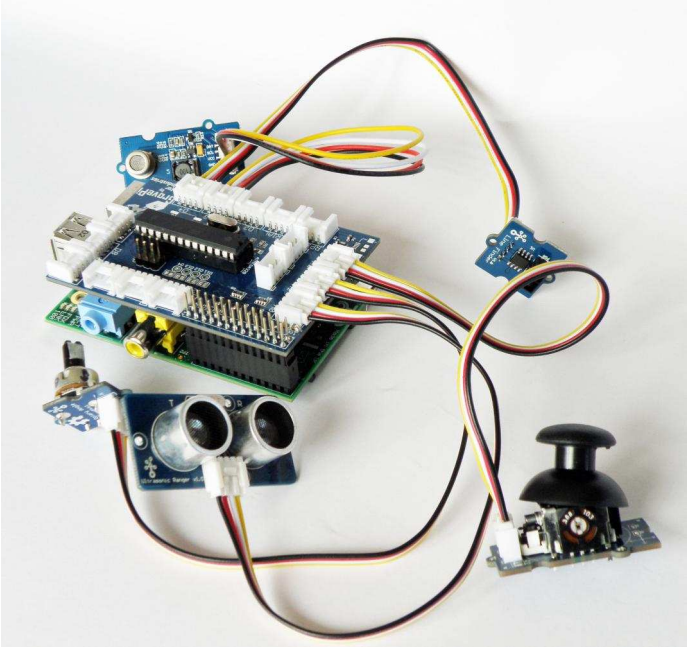


Getting Started

The GrovePi library can be downloaded from: https://github.com/DexterInd/GrovePi

The dependencies for the GrovePi can be installed using a script available in the ‘Script’ directory of the GrovePi library and the library could be used upon reboot.

This article discusses some simple Python3 examples that can be used with the GrovePi. The example code discussed is available at: https://github.com/yamanoorsai/GrovePi_Test



Blinking LED example

To test that the GrovePi is working correctly, a simple LED example can be used. The GrovePi is controlled from the Raspberry Pi using Python and the GrovePi module. To cause the LED on the GrovePi to blink:

```
import time
import grovepi

grovepi.pinMode(7,"OUTPUT")
time.sleep(1)

while True:
  try:
    grovepi.digitalWrite(7,1)
    time.sleep(1)
    grovepi.digitalWrite(7,0)
    time.sleep(1)
  except IOError:
    print("Error")
```

At the top of the program, the GrovePi module is imported. Then pin 7 is configured as an output pin.

```
    grovepi.pinMode(7,"OUTPUT")
```

Within the loop, the **digitalWrite** function is used to turn the LED on and off. The program waits for one second after turning the LED on and one second after turning the LED off.

The GrovePi function naming scheme is similar to the Arduino libraries and WiringPi.

Data-logger application

The GrovePi can be used to measure the air quality and log the associated data. This is achieved using an air quality sensor module and a real time clock (RTC) module,

```
tk = Tk()
# Add a canvas area ready for drawing on
SensorButton = Button(tk,text="Update Air
 Quality Value",command= Analog)
SensorButton.pack()
#Add an exit button
btn = Button(tk, text="Exit",
 command=terminate)
btn.pack()

Label1 = Label(tk)
Label1.pack()
Label1.configure(text=("Sensor Value = %d")%
 grovepi.analogRead(0))
Label2 = Label(tk)
Label2.pack()
```

In this example, the GrovePi is used to read the current time from the RTC module and display it in a desktop window. The sensor value and associated readout time is written into a text file.

This program makes use of the Tkinter graphical user interface (GUI) toolkit, to implement a simple user interface. A basic guide to using Tkinter from Python is given at:
http://pihw.wordpress.com/lessons/rgb-led-lesson-5-creating-a-graphical-user-interface/

The Tkinter object is initialised and two button widgets are added. One button widget called **SensorButton** is used to record the analog value from the air quality sensor along with the time read from the RTC module. A callback

function called **Analog** is registered to react to **SensorButton** changes.

```
def Analog():
  global Label1
  try:
    Value = (int)(grovepi.analogRead(0))
    DisplayTime = grovepi.rtc_getTime()
    Label1.configure(text=("Sensor value =
     %d")% Value)
    text_file = open("Output.txt", "a")
    text_file.write("%02d:%02d:%02d Sensor
     value:%d\n" % (DisplayTime[1],
     DisplayTime[2],DisplayTime[3],Value))
    text_file.close()
  except IOError:
    pass
```

The other button called **Exit** terminates the application and destroys the window.

```
def terminate():
  global tk
  tk.destroy()
```

We use the **Label** widget to update the time using the RTC module.

```
def Display_Time():
  global Label2,DisplayTime
  try:
    DisplayTime = grovepi.rtc_getTime()
    Label2.configure(text=("%02d:%02d:%02d")
     %(DisplayTime[1],DisplayTime[2],
     DisplayTime[3]))
    tk.after(1000,Display_Time)
  except Exception,e:
    print 'Exception was thrown', str(e)
    print "Error"
    tk.after(1000,Display_Time)
```

The function call **rtc_getTime()** returns an 8 byte array, containing the date and time information. This is set as the text in the **Label** widget. The text is updated every second, using the tk widgets method **after**.

```
    tk.after(1000,Display_Time)
```

Putting all the code together, the GUI looks something like this:

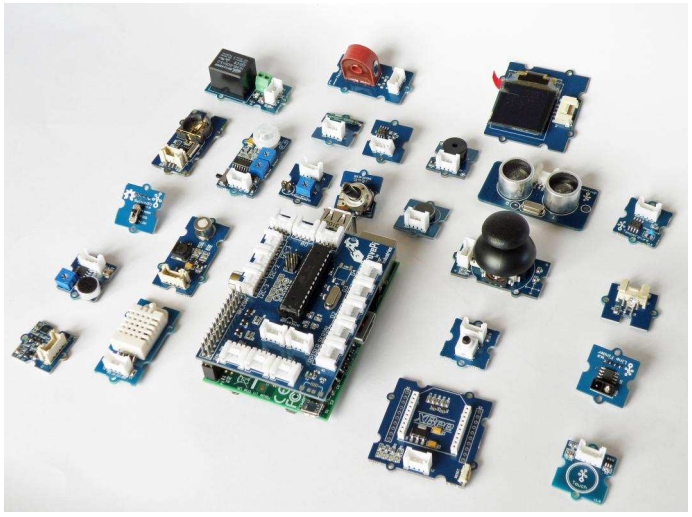
The application logs data as follows:

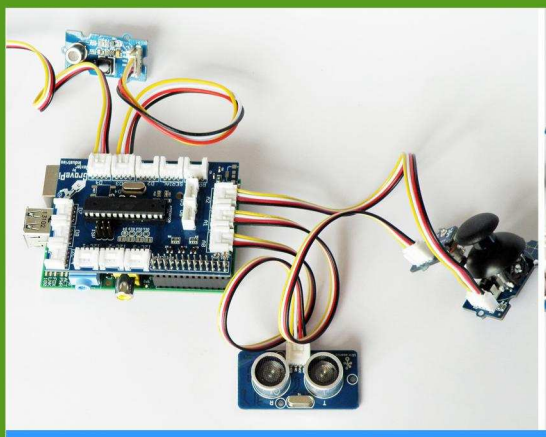
```
06:09:56 Sensor value: 260
```

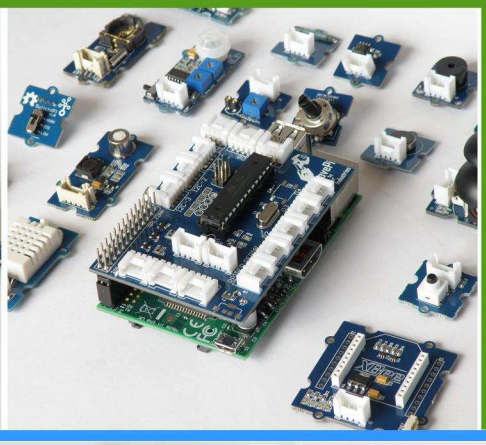
Conclusion

The GrovePi is a board that allows GrovePi sensors to be easily interface with the Raspberry Pi. The Grove Pi board costs 24USD. There are different grove modules, available from prices as low as 5USD (a relay module) to 50USD (an EMG detector used for detection of signals from skeletal muscles).

More examples for the Grove Pi are available at: http://www.dexterindustries.com/GrovePi/projects-for-the-raspberry-pi/





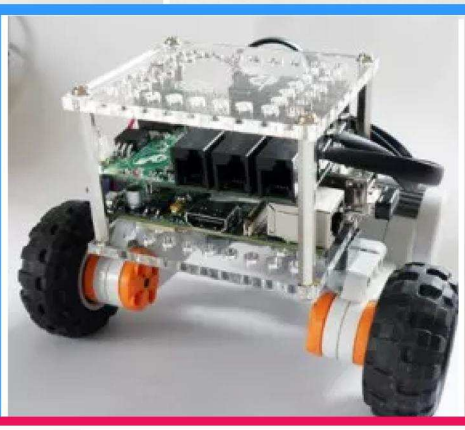


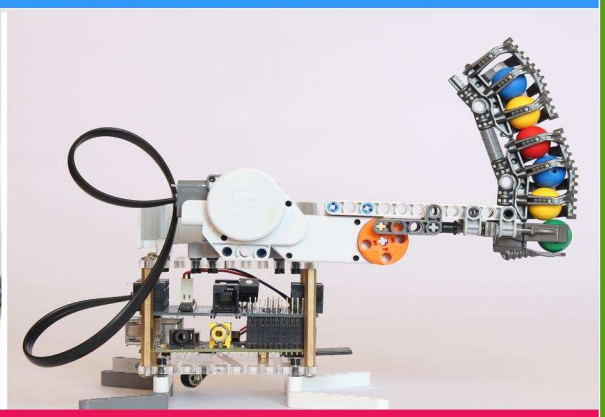
GrovePi

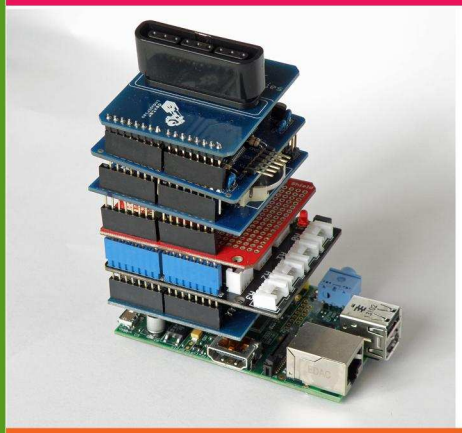
Connect Hundreds of Sensors to your Raspberry Pi

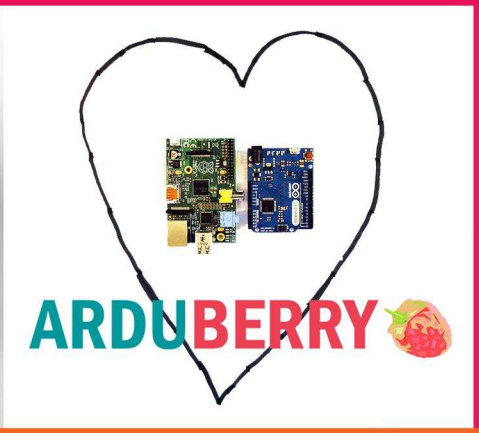
BrickPi

Turn your Raspberry Pi into a LEGO© Robot







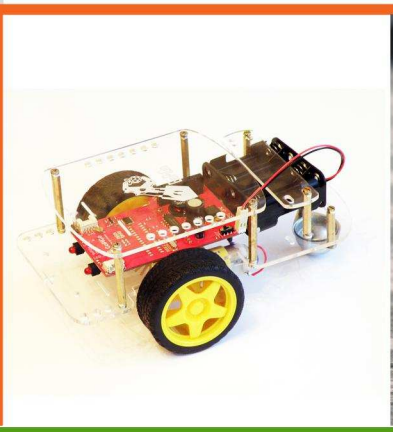


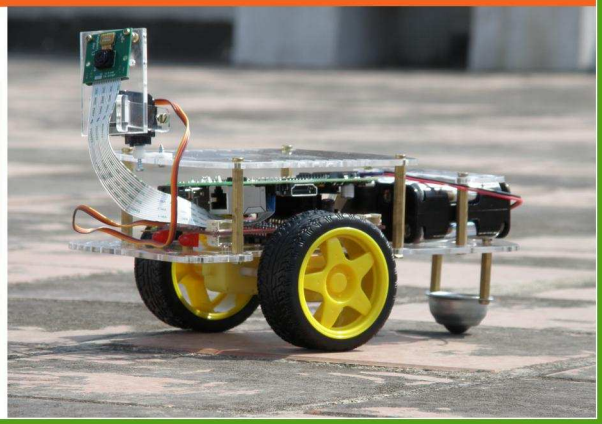
Arduberry

Unite the Raspberry Pi and Arduino

GoPiGo

Turn Your Raspberry Pi into a Robot

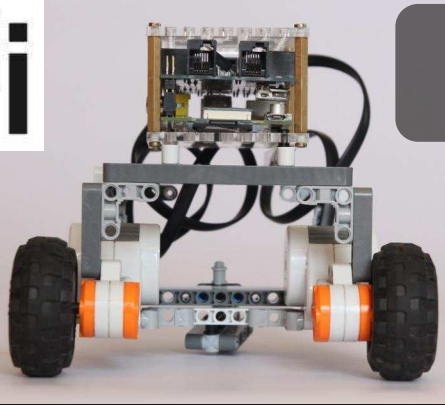




dexterindustries.com

MagPi Readers! Use the code “MagPi14” for a 10% discount in our store.

BrickPi



LEGO NXT INTERFACE
Plug in LEGO® sensors and motors

BrickPi Scratch interface with RpiScratchIO - part 3

SKILL LEVEL : INTERMEDIATE

W. H. Bell

MagPi Writer

Introduction

The BrickPi provides an interface between LEGO® MINDSTORMS® motors and sensors and the Raspberry Pi. The board has five sensor ports and four motor ports. The Raspberry Pi communicates with the BrickPi over the serial port (UART), which is available via pins 8 and 10 of the 26 pin header on the Raspberry Pi:
http://elinux.org/RPi_Low-level_peripherals

The BrickPi is able to read both the sensors and the motor encoder values, providing a simple interface to a range of MINDSTORMS® based projects. More details of the BrickPi are given in Issues 17 and 18 of The MagPi.

A local school decided to buy several BrickPi boards, for their engineering course. After a few tests, it was clear that a new Scratch driver was needed to allow the BrickPi to be used within the course material. To allow other Raspberry Pi GPIO connections to be used and provide easy configuration of the BrickPi, RpiScratchIO was chosen as the basis of the Scratch interface. More information on RpiScratchIO can be found in Issues 20 and 22 of The MagPi and at:
https://pypi.python.org/pypi/RpiScratchIO/

Installation & configuration

These installation instructions start from the basic Raspbian image that can be downloaded from the Raspberry Pi web site:
http://www.raspberrypi.org/downloads/

The BrickPi Python and Scratch interfaces are available as a Python module. To install both interfaces, together with their dependencies, open a terminal window and type:

```
sudo apt-get install -y python-serial \
python-setuptools python-dev
easy_install pip
pip install BrickPi
```

To use the serial port to communicate with the BrickPi, it has to be available as an input/output (I/O) connection. To allow a connection using the serial bus, the Raspbian configuration needs to be changed slightly. Type:

```
sudo -s
```

Then use the nano editor to edit `/boot/config.txt`:

```
nano /boot/config.txt
```

Go the end of the file and add:

```
init_uart_clock=32000000
```

Then save this file and open `/etc/inittab`. Find

```
T0:23
```

and put a # character in front, to comment it out. Then save it. Next, open `/boot/cmdline.txt` and remove

```
console=ttyAMA0,115200 kgdboc=ttyAMA0,115200
```

and save the file. Now reboot the Raspberry Pi:

```
reboot
```

Building the tank

Follow the instructions given at,
http://www.dexterindustries.com/BrickPi/projects/tank/
to build a tracked vehicle. Then put the BrickPi on top and connect the right motor to BrickPi port MA and the left motor to BrickPi port MB. Finally, add a forward-facing ultrasonic sensor and connect it to BrickPi sensor port S1. The BrickPi port labelling is given in The MagPi Issue 18 article on the BrickPi.

Human controlled tank

The BrickPi has to be appropriately configured for each different LEGO® sensor. Create a new file called `tank.cfg` that contains:

```
[DeviceTypes]
LEGO = import BrickPi; from BrickPi.BrickPiScratch import BrickPiScratch; BrickPiScratch()

[DeviceConnections]
LEGO = UART0

[BrickPi]
S1 = ULTRASONIC_CONT
MA =
MB =
```

Now open Scratch and enable remote sensor connections, by selecting the "Sensing" palette and right clicking on "sensor value" at the bottom of the tool palette. (For the current Raspbian version of Scratch, the remote sensor connections need to be disabled and then re-enabled.) Now start RpiScratchIO by typing:

```
RpiScratchIO tank.cfg
```

This creates new Scratch sensors with names within the ranges `LEGO:0-LEGO:3`, `LEGO:10-LEGO:13` and `LEGO:20-LEGO:23`, where `LEGO:0-LEGO:3` are the sensor ports S1-S4, `LEGO:10-LEGO:13` correspond to the values written to the four motor ports MA-MB and `LEGO:20-LEGO:23` are the motor encoders for MA-MB. The sensor (S1-S4) and the motor encoder (MA-MB) values can be updated in Scratch by broadcasting read commands. For example,

```
LEGO:read:0
```

transfers the current S1 value into the Scratch sensor `LEGO:0`. The motor speed can be changed by broadcasting a write command:

```
LEGO:write:10,255
```

where this sets the motor connected to MA to run forwards at full speed. To stop the motor attached to MA, use Scratch to broadcast:

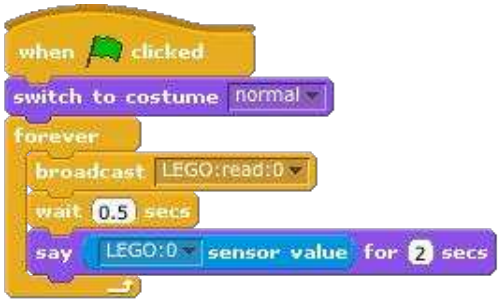
```
LEGO:write:10,0
```

If a sensor channel is used that has not been

configured in the configuration file, then a warning is reported. To update the sensors that are active in Scratch, exit RpiScratchIO by typing CTRL-C, edit the RpiScratchIO configuration file and then restart RpiScratchIO as before.

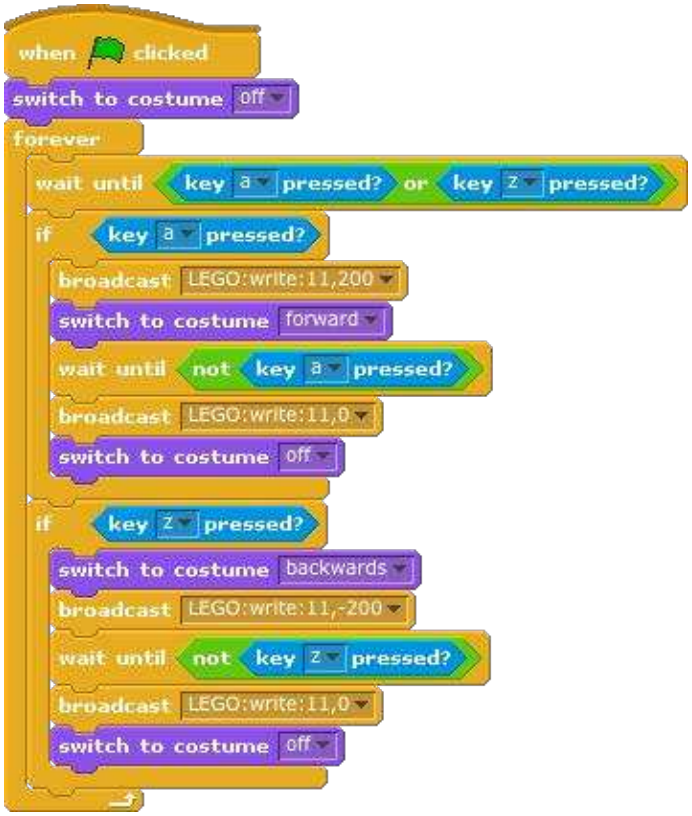
The BrickPiScratch interface can be easily used to control the position of the tank, using the keyboard. In this example, a simple top view of a tank was used as the main sprite.

Then two sprites were made, one for each track of the vehicle. The main sprite was then linked to the ultrasonic sensor and each track was linked to a motor. The main sprite program,



sends a read command to read the ultrasonic sensors, waits for half a second and then prints the value of the sensor. The value of the sensor is also used to change the costume of the main sprite, to indicate if there is an object close by.

The program for the left motor,



sets the motor to run forward when the *a* key is held down. When the key is released, the motor stops. To make the motor go backwards, the z key should be held down. When the motor is running forwards or backwards, the track sprite colour changes to show that the motor is running. The right track program is the same, but with the *d* and x keys and the motor channel 10.

Data acquisition & limits

To understand the limits of the Scratch driver, it is helpful to know what is happening within the driver and the BrickPi itself. The BrickPi Scratch diver is written in Python and communicates between Scratch and the BrickPi. If the Scratch driver sent the BrickPi a single motor controller command, the BrickPi would run the associated motor for about one second and then stop. This design choice was make in the Brick itself to prevent run-away robots. To make Scratch programs more efficient and simpler, the Scratch driver starts a data acquisition loop with the BrickPi that runs every tenth of a second. This loop sends the current motor values to the BrickPi and retrieves the current sensor values for the

sensors that are enabled in the configuration file. The current values for the sensors and motor encoders are stored in the BrickPi Scratch driver. In this manner, the Scratch code can send a single value to the scratch driver and see quick updates of motor values.

Scratch running on a Raspberry Pi cannot receive sensor updates faster than approximately half a second. While requesting a value and then receiving it completely decouples Scratch from the fast data acquisition loop inside the Scratch driver, it also uses more processor time. This is a good choice for monitoring long term changes, but does not work well for autonomous projects.

To reach the limit of Scratch I/O and performance, the BrickPi Scratch driver can be used to send regular sensor updates to Scratch. This has two implications: (1) reduced number of broadcast messages and (2) the ability to trigger Scratch from the automatic readout. Both of these changes ensure that the Scratch communication is as efficient as possible.

Autonomous tank

A human controlled vehicle is amusing, but a true robot should be able to function on its own. To make decisions, the sensor updates within Scratch have to be as fast as possible. This can be achieved by turning on the automatic update mechanism in the BrickPi Scratch driver. Copy the `tank.cfg` file and rename it as `autoTank.cfg`. Then change

```
BrickPiScratch()
```

to

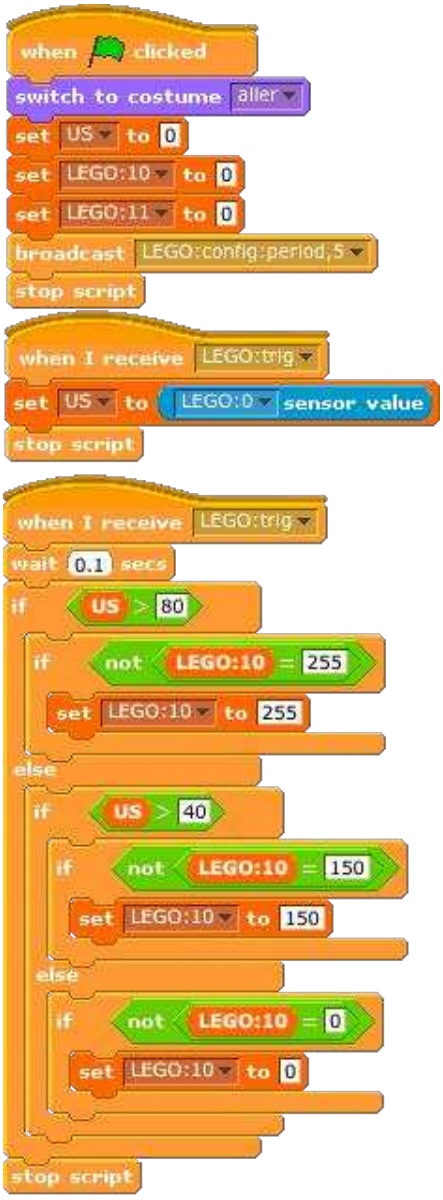
```
BrickPiScratch(0,"s")
```

The number zero is the period of the automatic update in units of tenths of a second, where zero disables the automatic update of sensor values. The value `"s"` configures all active sensors to be automatically updated. Other options are given at: https://pypi.python.org/pypi/BrickPi

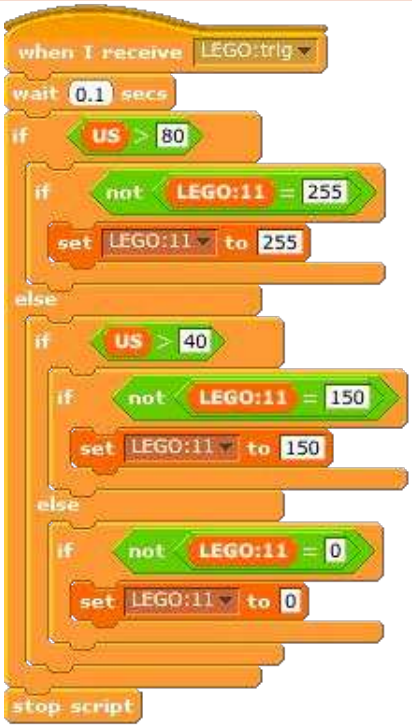
Close the previous Scratch window and open a new one. Set up the remote sensor connections as before. Then type,

```
RpiScratchIO autoTank.cfg
```

This time, the Scratch setup contains only one sprite. The program for the sprite is given below and on the next page.



When the green flag is clicked, the tank changes its costume to indicate that the automatic sensor update loop is running. The local ("For this sprite only") variable US and global ("For all sprites") variables `LEGO:10` and `LEGO:11` are all set to zero. Then the `config:period,5` command is sent to start the automatic updates, which then run once every half a second.



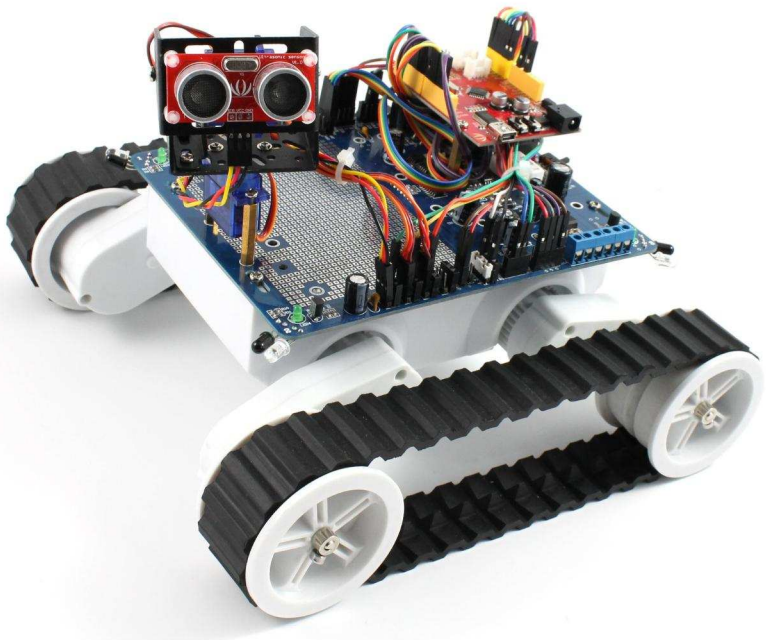
Each time the sensors are automatically updated, a `LEGO:trig` message is sent to Scratch. The `LEGO:trig` message is used to run the three sections that follow, where the first section stores the sensor value in the US variable and the other two sections control the two motors. When the `LEGO:10` or `LEGO:11` values are updated, they are automatically sent to the BrickPi channels MA and MB respectively.

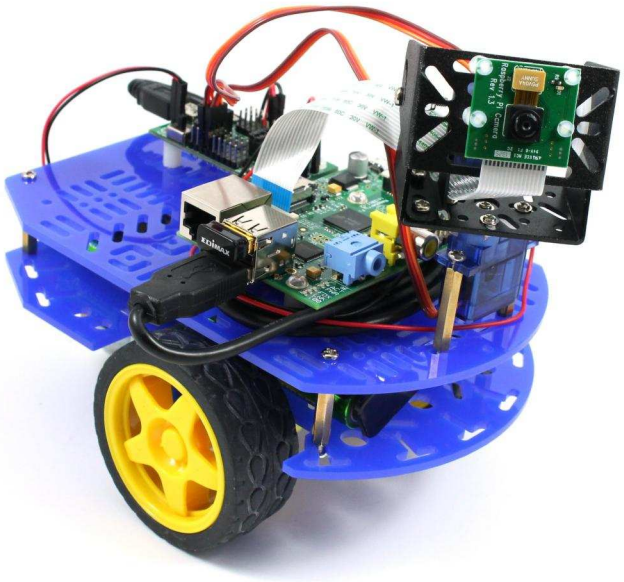
The last section is only run when the *q* key is pressed on the keyboard, which stops the automatic updates. The program then waits, in case one motor command is still being processed. Then the costume is set back to normal and both motors are stopped.

Try pointing the robot at a smooth wall and clicking on the green flag. When the wall is far away, the motors will run at high speed. Then the robot will slow down and finally stop.

LEGO® is a trademark of the LEGO Group of companies which does not sponsor, authorize or endorse this site

DAWN ROBOTICS





RASPBERRY PI AND ARDUINO ROBOT KITS

www.dawnrobotics.co.uk

METAL CASE FOR PI & CAMERA USE OUTSIDE!

DESIGNED FOR

TRI-POD MOUNT
THERMAL COOLING
REMOVABLE SIDE PANELS

AVAILABLE IN:
BLACK,
SILVER,
RED
& GREEN

BUY 2 AND
MIX & MATCH!

PiCE RASPBERRY PI
CAMERA ENCLOSURE

BY EDVENTURE www.ed-venture.biz

USE CODE: MAGPITHX23 FOR 10% DISCOUNT!

The MagPi's Second Birthday Competition

Over £2,000 worth of prizes to be won!

The MagPi has now reached its two year anniversary. With the support of many contributors from across the globe we have seen a wealth of interesting articles on our favourite tiny computer. To celebrate our second year we have a very big birthday competition for you all to enter, and all you have to do is answer the first five multiple choice questions below. The last two questions are optional, but we would love to hear your thoughts.

Enter at www.themagpi.com/birthday. All correct entries received by June 1st this year will be put into the prize draw, with the first 14 drawn winning a prize in the order listed. Winners will be notified by email and the winning names will be posted on The MagPi website.

All answers are to be found in The MagPi from issue 12 onwards. Full terms are available at: www.themagpi.com/birthday

Questions

Q1 Which game did we bring to Python written by Tim Hartnell?
- a) The Duke of Dragonfear
- b) Stronghold of the Dwarven Lords
- c) The Bannochburn Legacy

Q2 How many degrees north of the equator is the island of Curacao?
- a) 12 degrees
- b) 14 degrees
- c) 18 degrees

Q3 Which pirate symbol features on the chest of the BrickPi LEGO® man in The MagPi?
- a) Flintlock pistols
- b) Crossed cutlasses
- c) Skull and crossbones

Q4 Who was mentioned as having famously flicked paint off the end of a paint brush onto canvas?
- a) Jackson Pollock
- b) Andy Warhol
- c) Leonardo da Vinci

Q5 Who is Acorn's Elite commander?
- a) Commander Devereaux
- b) Commander Bluehair
- c) Commander Jameson

Q6 [Optional] What articles would you like to see in The MagPi over the next year?

Q7 [Opional] What do you use your Raspberry Pi(s) for?

Prizes

1st The MagPi Volumes 1 & 2, IO Pi 32, ADC Pi, RasPiO Breakout Pro, Bare Conductive House Kit, Wolfson Audio Card, BrickPi Starter Kit, GrovePi, Outdoor PiCE and weather shield, 1 year domain name and hosting for your Raspberry Pi Project, Laika Explorer Inventor Kit, RasPi Connect voucher, $100 Open Electrons voucher, Panavise 201, 312 & 371, UPiS Advanced with case and PiCoolFan, LEDBorg, PicoBorg, XLoBorg, TriBorg, Raspberry Pi Cookbook for Python Programmers & hardware kit, Adafruit Pi TFT & PiBow TFT case, Pi Supply Switch, RTK Motor controller board, Sweetbox with heatsinks ScorPi & CAMlot, Raspberry Pi Projects book, Adventures in Raspberry Pi book, FLIRC & remote, Raspberry Pi Mug Coaster.

2nd IO Pi 32, ADC Pi, Pi TFT, RasPiO Breakout Board, Bare Conductive House Kit, Wolfson Audio Card, Pi UPS, 1 year domain name and hosting for your Raspberry Pi project, Kano computer kit (eta July), RasPiConnect voucher, $75 Open Electrons voucher, LEDBorg, PicoBorg, Pi Supply Switch, Plusberry Pi Case (when available), Sweetbox with heatsinks & ScorPi, Raspberry Pi Projects for Dummies book, WiFi adaptor, Pibrella.

3rd Serial Pi RS232, RasPiO Port Reference Board, Bare Conductive Card Kit, Wolfson Audio Card, Pi UPS, RasPiConnect voucher, $75 Open Electrons voucher, LEDBorg, Pi Crust, Pi Supply Switch, Plusberry Pi Case (when available), ScorPi & CAMlot, Raspberry Pi for Dummies book, Tiny breadboard kit.

4th Serial Pi RS232, RasPiO Port Reference Board, Bare Conductive Card Kit, Pi RasPiConnect voucher, $50 Open Electrons voucher, UPiS Advanced with case & PiCoolFan, Pi Crust, Pi Supply Switch, Short Crust Case, Bright Pi, Plusberry Pi Case (when available), Sweetbox with heatsinks, Learning Python with Raspberry Pi book.

For the complete list visit www.themagpi.com/birthday

Sponsors

Pi Supply
KANO
OpenElectrons.com
Kitronik
PiBorg
Pimoroni
AB electronics UK
PANAVISE Innovative Holding Solutions
MiloCreek Where Innovation Flows
RasPi.TV
BARE CONDUCTIVE
EDVENTURE ED-VENTURE.BIZ
WILEY
adafruit INDUSTRIES
CW2. electronics
Ryanteck LTD
πmodules www.pimodules.com
The PiHut
dexter industries
Sweetbox by Graspinghand
Ethical WebSites™

Interfacing Minecraft with PiFace Digital

W. H. Bell
MagPi Writer

SKILL LEVEL : INTERMEDIATE

Minecraft is a very popular game that has more recently been used as a teaching tool, to encourage programming. The Raspberry Pi version of Minecraft is available for free. The game is packaged with a Python application programmer interface (API), which allows interactions with players and blocks. An introduction to this API was given in Issue 11 of The MagPi. There are also additional teaching resources, such as Craig Richardson's "Python Programming for Raspberry Pi" book.

The interaction with Minecraft via the Python API is not limited to just software. Input/output (I/O) devices that are connected to a Raspberry Pi can react to, or produce events in Minecraft. This opens up a range of possibilities. For example, the garage door could open when the Minecraft player goes into the garage or the doorbell could be connected to a Minecraft chat message, etc. Since the Python API sends commands to the Minecraft server process over a network, the Raspberry Pi that is interacting with the Minecraft session could be in the garden monitoring the weather.

In this article, the Minecraft API is used with the PiFace Digital expansion board to create some traps around the selected player.

PiFace Digital

The PiFace Digital expansion board provides a safe way to connect digital devices to the Raspberry Pi, via buffered screw terminals.

The board includes two relays that are suitable for low voltage applications, four switches, eight digital inputs, eight open collector outputs and eight LED indicators. The Raspberry Pi communicates with the PiFace via the SPI bus on the 26 pin header. There are Python and Scratch interfaces that are packaged for easy installation using the Debian package manager. More information on the PiFace can be found at: http://www.piface.org.uk/products/piface_digital/

PiFace Python interface

Starting from a recent Raspbian image, make sure that the Raspbian installation is up to date:

```
sudo apt-get update
sudo apt-get upgrade -y
```

Then start the raspi-config,

```
sudo raspi-config
```

Select the "Advanced Options" and choose "SPI". Then set the value to "Yes", select "Ok" and "Finish". (The Python library python-pifacedigitalio is already installed in the latest Raspbian image configuration.) PiFace example Python programs can be found in:

```
/usr/share/doc/python-pifacedigitalio/examples/
```

Installing Minecraft

If Minecraft is not already installed, then type:

```
cd $HOME
wget https://s3.amazonaws.com/assets.minecraft
.net/pi/minecraft-pi-0.1.1.tar.gz
tar zxvf minecraft-pi-0.1.1.tar.gz
```

Rather than copying the API files, the directory where the Python API is can be added to the PYTHONPATH variable. Use nano to edit the .bashrc file:

```
nano ~/.bashrc
```

Go to the end of the file and add:

```
# For Minecraft
MCPI_PY=$HOME/mcpi/api/python
if [[ -z $PYTHONPATH ]]; then
 export PYTHONPATH=$MCPI_PY
elif [[ $PYTHONPATH != *"$MCPI_PY"* ]]; then
  export PYTHONPATH="$PYTHONPATH:$MCPI_PY"
fi
unset MCPI_PY
```

Then save the file.

Minecraft traps

There are many actions that could be triggered by hardware input changes. In this article, some dramatic events that are centred on a given player are used.

Create a new file called McTraps.py in the present working directory, add the Python given at the bottom of this page and save the file. Then open a second file called mcControl.py and add the Python found on the next page.

```
import mcpi
from mcpi.block import *
import time

def sandTrap(mc):
  pos = mc.player.getTilePos()
  mc.setBlocks(pos.x-10,pos.y+15,pos.z-10,pos.x+10,pos.y+18,pos.z+10,SAND)
  mc.postToChat("Welcome to the beach!")

def volcanoTrap(mc):
  pos = mc.player.getTilePos()
  mc.postToChat("Warning.. volcano!")
  time.sleep(1)
  mc.setBlocks(pos.x,pos.y-50,pos.z,pos.x,pos.y-1,pos.z,LAVA)
  time.sleep(1)
  mc.setBlocks(pos.x-2,pos.y,pos.z-2,pos.x+2,pos.y+2,pos.z+2,LAVA)
  mc.postToChat("A bit too hot!")

def holeTrap(mc):
  pos = mc.player.getTilePos()
  mc.postToChat("Watch your feet!")
  time.sleep(1)
  mc.setBlocks(pos.x-2,pos.y-40,pos.z-2,pos.x+2,pos.y,pos.z+2,AIR)
```

```
#!/usr/bin/env python
import mcpi
from mcpi.minecraft import Minecraft
import pifacedigitalio
from McTraps import *
import sys,time

class McControl:
  def __init__(self, ips):
    self.ips = []
    self.ips += ips

    # Open connections with the Minecraft sessions
    self.connections={}
    for ip in self.ips:
      try:
        #self.connections[ip] = Minecraft.create(ip)
        self.connections[ip] = None
      except:
        print("ERROR: cannot connect to Minecraft on %s" % ip)
        sys.exit(1)

    # Store the number of connections and initialise the current connection to be the first one
    self.connectionNumber = 0
    self.numberOfConnection = len(ips)

    # Setup an input event listener, one per switch on the PiFace
    pifacedigital = pifacedigitalio.PiFaceDigital()
    self.listener = pifacedigitalio.InputEventListener(chip=pifacedigital)
    for i in range(4):
      self.listener.register(i, pifacedigitalio.IODIR_ON, self.switchPressed)

  def listen(self):
    # Start listening to the PiFace
    self.listener.activate()
    print(">> Listening to PiFace")

  def shutdown(self):
    # Stop listening to the PiFace
    self.listener.deactivate()
    print(">> Listeners shutdown")

  def nextConnection(self):
    # Change to the connection associated with the next IP in the list.
    self.connectionNumber  = self.connectionNumber  + 1
    if self.connectionNumber  >= self.numberOfConnection:
      self.connectionNumber  = 0
    print(">> Using connection to %s" % self.ips[self.connectionNumber])

  def getConnection(self):
    # Return the connection associated with the selected IP
    if not self.ips[self.connectionNumber] in self.connections.keys():
      raise Exception("Error: no connection to %s" % self.ips[self.connectionNumber])
    return self.connections[self.ips[self.connectionNumber]]

  def switchPressed(self,event):
    # Handle switch press events:
    # (0) - If the first switch has been pressed, change to the next IP
    if  event.pin_num == 0:
      self.nextConnection()
      return None

    # Get the current Minecraft connection
    mc = self.getConnection()
```

```
    # (1-3) - Use the switch number to decide which trap to run
    if event.pin_num == 1:
      print(">> Sand trap")
      sandTrap(mc)
    elif event.pin_num == 2:
      print(">> Volcano trap")
      volcanoTrap(mc)
    elif event.pin_num == 3:
      print(">> Hole trap")
      holeTrap(mc)
    else:
      raise Exception("ERROR: pin number is out of range.")

if __name__ == "__main__":
  # If no command line options are provide, assume that the localhost is running Minecraft
  ips = []
  if len(sys.argv) == 1:
    ips += ["localhost"]
  else:
    ips += sys.argv[1:]

  # Start the MineCraft connections and PiFace listeners.
  # The listeners are shutdown went the program exits.
  mcControl = McControl(ips)
  try:
    mcControl.listen()
    while(1):
      time.sleep(1000)
  except KeyboardInterrupt:
    mcControl.shutdown()
```

Set the mcControl.py file as executable,

```
chmod 755 mcControl.py
```

Then use a new terminal window to start Minecraft, using the local Raspberry Pi:

```
cd mcpi
./minecraft-pi
```

Once Minecraft is running and a world has been created. Click on the tab key, to break the window focus. Then select the original terminal window and type:

```
./mcControl.py
```

By default, the program uses the localhost. To access one or more remote Minecraft games, type:

```
./mcControl.py 192.168.1.20 192.168.1.21
```

etc., where the IP addresses are the IP addresses of the Raspberry Pis running Minecraft. The IP address on a Raspberry Pi can be found by typing

```
ifconfig
```

in a terminal window.

Once the mcControl.py program is running, pressing switch 0 on the PiFace will cause the program to change the connection used for the tricks. This is useful, when more than one Minecraft session is in the connection dictionary. Buttons 1, 2 and 3, run the sandTrap, volcanoTrap and holeTrap functions respectively.

To prevent a lot of CPU being used, the McControl class creates a InputEventListener object. This listener is then used to associate the switchPressed function with one of the four switches being pressed. Once the listener.activate() function has been called, the program continues to listen for PiFace switch changes until it is closed with CTRL-C.

VISIT US ON

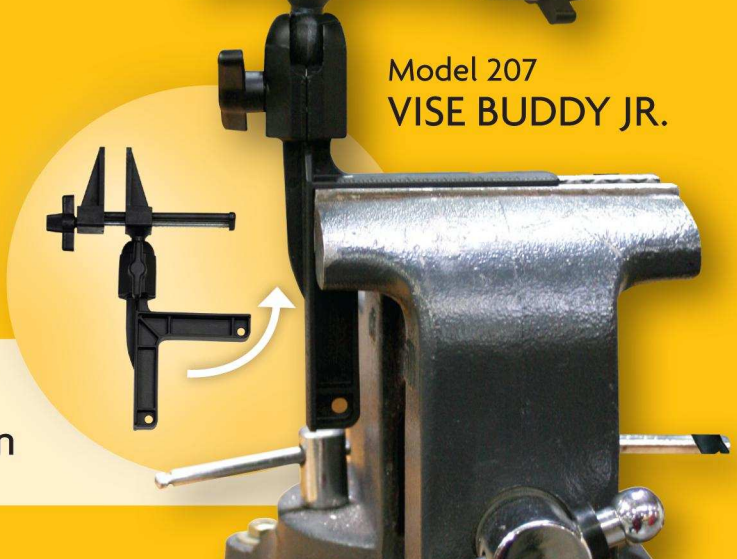
WORLD'S MOST VERSATILE CIRCUIT BOARD HOLDERS

Model 209
VACUUM
BASE PV JR.

Model 201
PV JR.

- Work-holding tools for electronics projects
- Circuit board holders make soldering easy & fun
- Versatile hobby vises for any project

Model 207
VISE BUDDY JR.

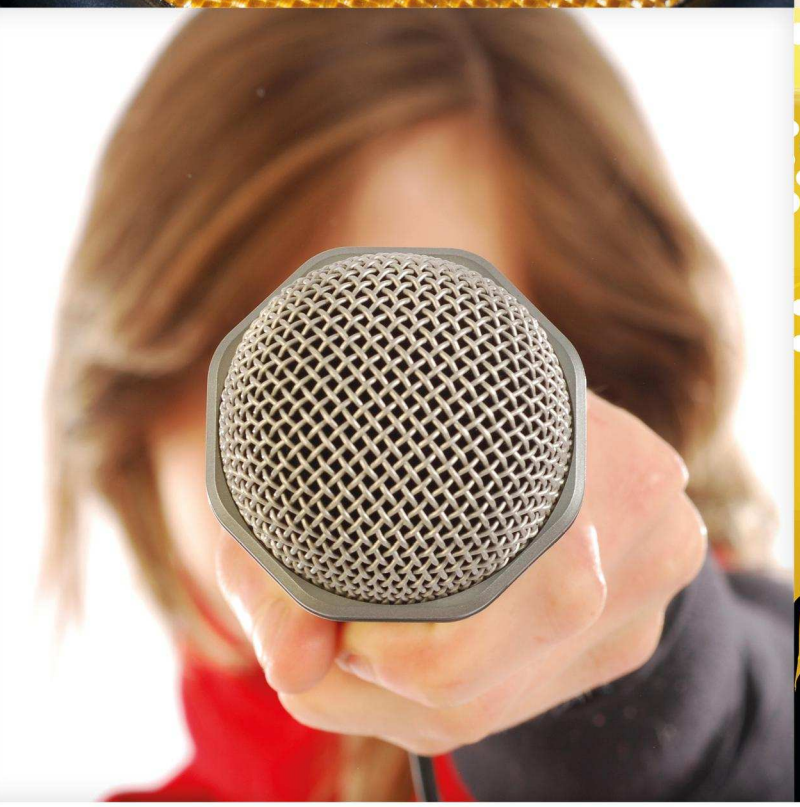


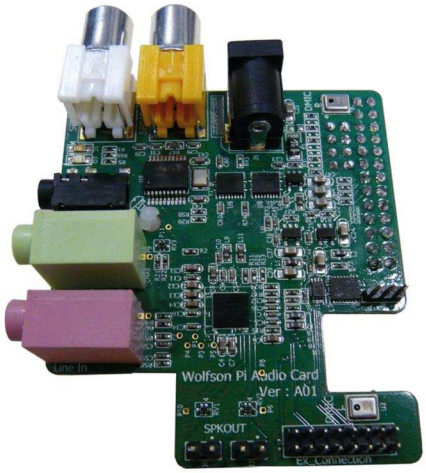
PANAVISE®
Innovative Holding Solutions

www.panavise.com

PIMORONI

PANAVISE IS AVAILABLE AT
http://shop.pimoroni.com





Wolfson Audio Card

High Definition audio for the Raspberry Pi®

For more information on the Wolfson Audio Card and many other exclusive accessories for the Raspberry Pi visit **www.element14.com/raspberrypi**

Exclusively from element14

® Raspberry Pi is a trademark of the Raspberry Pi Foundation. © Farnell element14 All Rights Reserved.

wolfson™ AUDIO

GPIO Sensing: Using 1-Wire temperature sensors - Part 2

Jacob Marsh

ModMyPi

SKILL LEVEL : BEGINNER

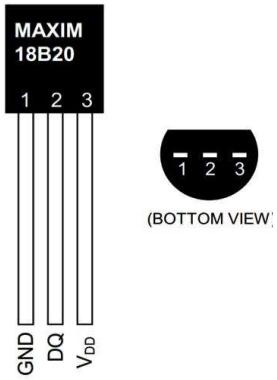
1-Wire sensors

In previous tutorials we've outlined the integration of simple sensors and switches with the Raspberry Pi. These components have had a simple on/off or high/low output, which is sensed by the Raspberry Pi. Our PIR movement sensor tutorial in Issue 21, for example, simply says "Yes, I've detected movement".

So, what happens when we connect a more advanced sensor and want to read more complex data? In this tutorial we will connect a 1-Wire digital thermometer sensor and programme our Raspberry Pi to read the output of the temperature it senses!

In 1-Wire sensors all data is sent down one wire, which makes it great for microcontrollers and computers, such as the Raspberry Pi, as it only requires one GPIO pin for sensing. In addition to this, most 1-Wire sensors will come with a unique serial code (more on this later) which means you can connect multiple units without them interfering with each other.

The sensor we're going to use in this tutorial is the Maxim DS18B20+ Programmable Resolution 1-Wire Digital Thermometer. The DS18B20+ has a similar layout to transistors, called the TO-92 package, with three pins: GND, Data (DQ) and 3.3V power line (V_{DD}). You also need some jumper wires, a breadboard and a 4.7kΩ (or 10kΩ) resistor.

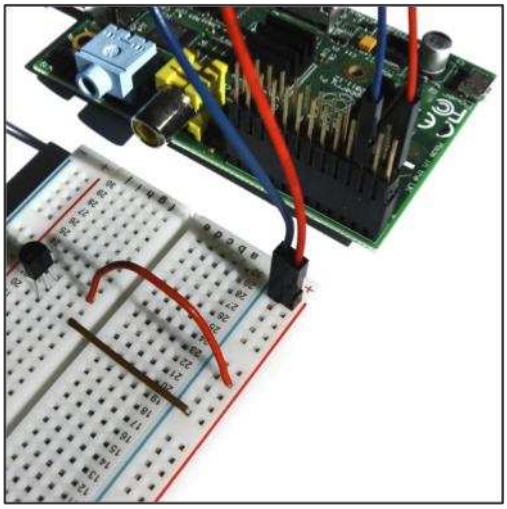


The resistor in this setup is used as a 'pull-up' for the data-line, and should be connected between the DQ and V_{DD} line. It ensures that the 1-Wire data line is at a defined logic level and limits interference from electrical noise if our pin was left floating. We are also going to use GPIO 4 [Pin 7] as the driver pin for sensing the thermometer output. This is the dedicated pin for 1-Wire GPIO sensing.

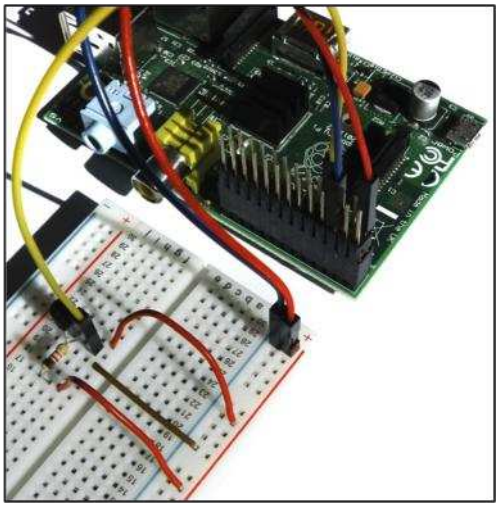
Hooking it up

1. Connect GPIO GND [Pin 6] on the Raspberry Pi to the negative rail on the breadboard.
2. Connect GPIO 3.3V [Pin 1] on the Raspberry Pi to the positive rail on the breadboard.
3. Plug the DS18B20+ into your breadboard,

ensuring that all three pins are in different rows. Familiarise yourself with the pin layout, as it is quite easy to hook it up backwards!
4. Connect DS18B20+ GND [Pin 1] to the negative rail of the breadboard.
5. Connect DS18B20+ V_{DD} [Pin 3] to the positive rail of the breadboard.



6. Place your 4.7kΩ resistor between DS18B20+ DQ [Pin 2] and a free row on your breadboard.
7. Connect that free end of the 4.7kΩ resistor to the positive rail of the breadboard.
8. Finally, connect DS18B20+ DQ [Pin 2] to GPIO 4 [Pin 7] with a jumper wire.



That's it! We are now ready for some programming!

Programming

With a little set up, the DS18B20+ can be read directly from the command line without the need for any Python programming. However, this requires us to input a command every time we want to know the temperature reading. In order to introduce some concepts for 1-Wire interfacing, we will access it via the command line first and then we will write a Python program which will read the temperature automatically at set time intervals.

The Raspberry Pi comes equipped with a range of drivers for interfacing. However, it's not feasible to load every driver when the system boots, as it increases the boot time significantly and uses a considerable amount of system resources for redundant processes. These drivers are therefore stored as loadable modules and the command `modprobe` is employed to boot them into the Linux kernel when they're required.

The following two commands load the 1-Wire and thermometer drivers on GPIO 4. At the command line enter:

```
sudo modprobe w1-gpio
sudo modprobe w1-therm
```

We then need to change directory to our 1-Wire device folder and list the devices in order to ensure that our thermometer has loaded correctly. Enter:

```
cd /sys/bus/w1/devices
ls
```

In the device list, your sensor should be listed as a series of numbers and letters. In my case, the device is registered as 28-000005e2fdc3. You then need to access the sensor with the `cd` command, replacing the serial number with that from your own sensor. Enter:

```
cd 28-000005e2fdc3
```

The sensor periodically writes to the `w1_slave` file. We can use the `cat` command to read it:

```
cat w1_slave
```

This yields the following two lines of text, with the output `t` showing the temperature in milli-degrees Celsius. Divide this number by 1000 to get the temperature in degrees, e.g. the temperature reading we've received is 23.125 degrees Celsius.

```
72 01 4b 46 7f ff 0e 10 57 : crc=57 YES
72 01 4b 46 7f ff 0e 10 57 t=23125
```

In terms of reading from the sensor, this is all that's required from the command line. Try holding onto the thermometer for a few seconds and then take another reading. Spot the increase? With these commands in mind, we can now write a Python program to output our temperature data automatically.

Python program

Our first step is to import the required modules. The os module allows us to enable our 1-Wire drivers and interface with the sensor. The time module allows the Raspberry Pi to define time, and enables the use of time periods in our code.

```
import os
import time
```

We then need to load our drivers:

```
os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')
```

The next step is to define the sensor's output file (the `w1_slave` file) as defined above. Remember to utilise your own temperature sensor's serial code!

```
temp_sensor = '/sys/bus/w1/devices/28-000005e2
fdc3/w1_slave'
```

We then need to define a variable for our raw temperature value, `temp_raw`; the two lines output by the sensor, as demonstrated by the command line example. We could simply print this statement now, however we are going to process it into something more usable. To do this we open, read, record and then close the `temp_sensor` file. We use the `return` function here, in order to recall this data at a later stage in our code.

```
def temp_raw():
  f = open(temp_sensor, 'r')
  lines = f.readlines()
  f.close()
  return lines
```

First, we check our variable from the previous function for any errors. If you study our original output, as shown in the command line example, we get two lines of output code. The first line was "72 01 4b 46 7f ff 0e 10 57 : crc=57 YES". We strip this line, except for the last three characters, and check for the "YES" signal, which indicates a successful temperature reading from the sensor. In Python, not-equal is defined as "!=", so here we are saying that while the reading does not equal "YES", sleep for 0.2s and repeat.

```
def read_temp():
  lines = temp_raw()
  while lines[0].strip()[-3:] != 'YES':
    time.sleep(0.2)
    lines = temp_raw()
```

Once a YES signal has been received, we proceed to our second line of output code. In our example this was "72 01 4b 46 7f ff 0e 10 57 t=23125". We find our temperature output "t=", check it for errors and strip the output of the "t=" phrase to leave just the temperature data. Finally we run two calculations to give us the temperature in Celsius and Fahrenheit.

```
  temp_output = lines[1].find('t=')
  if temp_output != -1:
    temp_string = lines[1].strip()[temp_output
+2:]
    temp_c = float(temp_string) / 1000.0
    temp_f = temp_c * 9.0 / 5.0 + 32.0
    return temp_c, temp_f
```

Finally, we loop our process and tell it to output our temperature data every 1 second.

```
while True:
  print(read_temp())
  time.sleep(1)
```

That's our code! A screenshot of the complete program is shown below. Save your program and run it to display the temperature output, as shown on the right.

```
pi@raspberrypi ~ $ sudo python temp_2.py
(23.437, 74.1866)
(23.437, 74.1866)
(23.437, 74.1866)
(23.437, 74.1866)
(23.437, 74.1866)
(25.5, 77.9)
(27.25, 81.05)
(28.312, 82.9616)
(29.0, 84.2)
(29.437, 84.9866)
(29.75, 85.55)
(29.687, 85.4366)
(29.0, 84.2)
(28.25, 82.85)
```

Multiple sensors

DS18B20+ sensors can be connected in parallel and accessed using their unique serial number. Our Python example can be edited to access and read from multiple sensors!

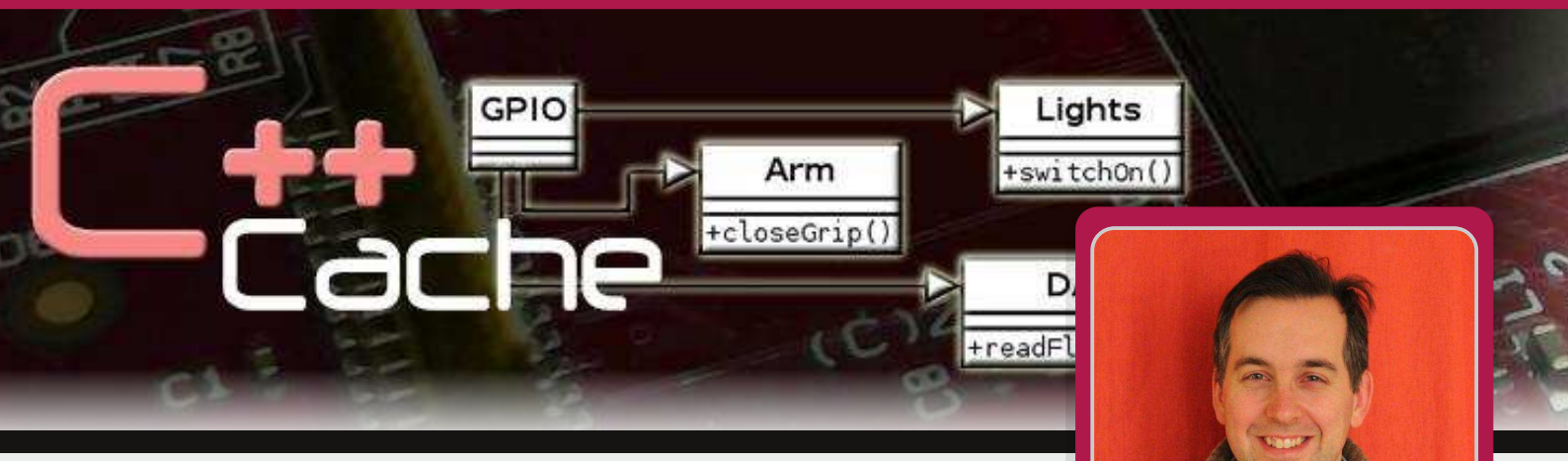
As always, the DS18B20+ sensor and all components are available separately or as part of our workshop kit from the ModMyPi website http://www.modmypi.com.

```
import os
import time

os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')

temp_sensor = '/sys/bus/w1/devices/28-000005e2fdc3/w1_slave'

def temp_raw():
    f = open(temp_sensor, 'r')
    lines = f.readlines()
    f.close()
    return lines

def read_temp():
    lines = temp_raw()
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = temp_raw()
    temp_output = lines[1].find('t=')
    if temp_output != -1:
        temp_string = lines[1].strip()[temp_output+2:]
        temp_c = float(temp_string) / 1000.0
        temp_f = temp_c * 9.0 / 5.0 + 32.0
        return temp_c, temp_f

while True:
        print(read_temp())
        time.sleep(1)
```

MODMYPI

This article is sponsored by ModMyPi

All breakout boards and accessories used in this tutorial are available for worldwide shipping from the ModMyPi webshop at www.modmypi.com



5 - Classes

W. H. Bell

MagPi Writer

SKILL LEVEL : ADVANCED

Welcome back to the C++ Cache. The subject of this month's article is the introduction of C++ classes. Before continuing, it may be helpful to read previous C articles in Issues 3, 4, 5, 6, 9, 13 and 17, and previous C++ articles in Issues 7, 8, 10 and 18.

Object-orientated programming

When programs become very large and there are many data structures associated with many different pieces of a program, then it can be helpful to associate particular functions with particular data structures. In object-orientated programming, the basic building block of a program is an object. An object can include functions and data members. Object-orientated programming represents a different style of programming, where concepts are grouped together to match purposes. For example, a car object could contain some variables such as the number of seats or the petrol left in the tank. The car object could also contain a GPS function that would return the position of the car in space.

Each object is instantiated in a similar manner as a simple variable. Instead of a simple type, an object is instantiated with a class. In the case of a simple variable,

```
int i;
```

the type is `int` and the variable is `i`. An object can be instantiated using the `Square` class in a similar way,

```
Square s;
```

Just as two `int` variables do not share the same memory location by default, two objects also do not share the same memory location by default. This means that if a value inside an object is changed, it will not normally affect the values within another object of the same class. When an object is instantiated, the associated constructor function described in the class declaration is called. The constructor function is typically used to initialise data members that belong to the class.

A first C++ class

Open a terminal window. Then use a text editor, such as nano or emacs, to create a file called `square.h`. Copy the code below into `square.h` and save the file.

```
#ifndef SQUARE_H
#define SQUARE_H

class Square {
public:
  Square(void);  // Default constructor
  Square(double side, char colour);  // Constructor with arguments
  double area();  // Return the area of a square
  double colour() { return m_colour; }  // The colour character

private:
  double m_side; // The length of a side
  char m_colour; // The colour character
};

#endif
```

The `square.h` header file contains the class declaration, where name of the class is `Square`. The `Square` class contains two constructor functions, two other member functions, and two data members. The name of the constructor function must be the same as the name of the class. The program uses the first constructor if no arguments are given and the second constructor if two values are provided. If the object instantiation does not match either of these constructors, then the compiler will report an error.

Headed files support limited functionality. For example, it is possible to return a value from function that is declared in a header file. In this case, the `colour()` function is declared and implemented in the header file. In contrast, the `area()` function is declared in the header file, but is not implemented in the header file. In this example, the implementation of the constructors and the `area()` function is given in a `.cpp` source file.

Within a class declaration, functions and data members can be `public`, `protected` or `private`. `public` members can be accessed from outside the class, `protected` members can only be accessed by objects of the same class or of a derived class and `private` members can only be accessed by objects of the same class. (The usefulness of `protected` members becomes clear once inheritance has been introduced.) In the case of `Square`, all of the functions are `public` and both data members are `private`.

The implementation of a class is given in a `.cpp` (or `.cc` or `.cxx` or `.C`) file. Create a file called `Square.cpp`. Then add the code found at the top of the next page and save the file.

Since the `Square.cpp` file includes the implementation of functions that were defined in the class declaration, it has to include the class declaration itself. This is included by including the header file `Square.h`. The `cmath` header file is needed to use the pow function, which is used to square the `side` value.

Below the include statements, the `Square.cpp` file contains the implementation of the three functions that were defined in the header file but were not implemented in the header file. Each of the function names is prefixed by the class name and two colons. The constructor functions do not have a declared return type, since they return

```
#include "Square.h"
#include <cmath>

Square::Square(void):
  m_side(0.),
  m_colour('0') {
}

Square::Square(double side, char colour):
  m_side(side),
  m_colour(colour) {
}

double Square::area() {
  return std::pow(m_side,2);
}
```

an object of the given class. For other functions, the return type must be given to the left of the class name. In this example, the type `double` is given before `Square::area()`. In the constructors, values can be assigned to data members using the parentheses notation given or with the assignment operator (=). The data members act as global variables within the functions of the class. Since they are both `private` data members, they are commonly prefixed with `m_`. This is not a requirement, but a convention that makes reading C++ code a little simpler.

The last part of this introduction is the use of the class `Square`. Create a new file called `main.cpp`. Then add the code below and save the file.

```
#include <iostream>
#include "Square.h"

using namespace std;

int main() {
  Square s; // Using the default constructor
  Square s2(3.0,'b'); // Using the second constructor

  cout << "s.area()=" << s.area() << ", s.colour()=" << s.colour() << endl;
  cout << "s2.area()=" << s2.area() << ", s2.colour()=" << s2.colour() << endl;

  return 0;
}
```

To use the class, the `main.cpp` file includes the header file `Square.h`. Two objects are instantiated, where the first instantiation calls the default constructor and the second instantiation calls the second constructor. Then the two functions `area()` and `colour()` are called and the values of the area and the colour (integer value) is printed on the screen. When calling a function of a given object, the function name is prefixed with the object name. In this example, the objects are created on the stack and go out of scope in the same way as a simple variable. This means that the two objects are destroyed when the `main()` function finishes.

Rather than type g++ several times, a `Makefile` can be used to compile the two `.cpp` files and produce an executable. Create a new file called `Makefile`, add the code at the top of the next page and save the file. The indented lines should be indented by a single tab, rather than spaces. (More information on Makefiles is provided

```
CC=g++
TARGET=square
OBJECTS=main.o Square.o

$(TARGET): $(OBJECTS)
  @echo "** Linking Executable"
  $(CC) $(OBJECTS) -o $(TARGET)

clean:
  @rm -f *.o *~

veryclean: clean
  @rm -f $(TARGET)

%.o: %.cpp
  @echo "** Compiling C++ Source"
  $(CC) -c $(INCFLAGS) $<
```

in Issue 7 of The MagPi.) The Makefile should be in the same directory as the `Square.h`, `Square.cpp` and `main.cpp` files. Then type

```
make
```

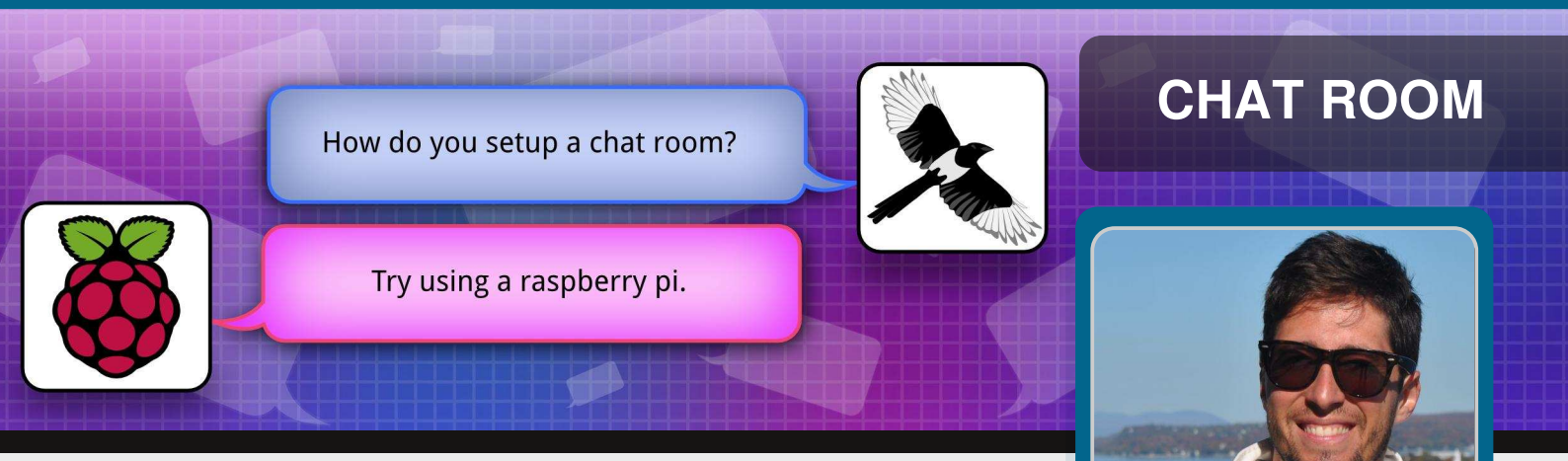
to compile the code and build the executable. Once the executable has been build, run the program:

```
./square
```

Objects on the heap

When an object is needed within several different functions calls, it might be helpful to create it on the heap instead. The difference between creating an object on the stack and the heap is that objects on the stack are automatically cleaned up when they go out of scope, whereas objects on the heap stay in memory. To clean up an object on the heap, it has to be explicitly deleted. A modified version of main.cpp is given below, where the objects are created on the heap instead. The syntax `s->colour()` is short hand for `(*s).colour()`.

```
#include <iostream>
#include "Square.h"

using namespace std;

int main() {
  Square *s = new Square(); // Using the default constructor
  Square *s2 = new Square(3.0,'b'); // Using the second constructor

  cout << "s->area()=" << s->area() << ", s->colour()=" << s->colour() << endl;
  cout << "s2->area()=" << s2->area() << ", s2->colour()=" << s2->colour() << endl;

  delete s;
  delete s2;

  return 0;
}
```



Turn your Raspberry Pi into an XMPP Chat Server

Gianluca Serra

Guest Writer

SKILL LEVEL : INTERMEDIATE

The storyDue to its low cost and power consumption, the Raspberry Pi is probably the best device to build a home server from. A home server is able to do a lot of useful things, including running as a media center, home repository, power or temperature monitoring and much more.

In my opinion, one of the more interesting uses of the Raspberry Pi is as an XMPP server. XMPP is a standard protocol for Instant Messaging, used by GTalk for example, that provides infrastructure for an XMPP server to send messages and two or more XMPP clients to exchange messages to each other.

I was searching for the best XMPP server to put on a Raspberry Pi when I came across Prosody (https://prosody.im). Prosody is a lightweight XMPP server written in LUA (a fast programming language based on C and used for game programming) with all of the basic capabilities of an XMPP server, plus some extra-modules you can add as you wish, such as an admin console, secure communication, etc.

With a XMPP server installed on your Raspberry Pi you can create a chat server to be used on your intranet Wi-Fi network (between you and your relatives, for example) or a real Internet chat server (like GTalk or WhatsApp) exposed through the Internet and under your control.

In this article we'll describe how to create an Internet chat server with private and restricted access.

Setup Prosody

Prosody has another big advantage: it is in the Raspbian repository. You can install it simply by typing:

```
$ sudo apt-get update
$ sudo apt-get install prosody
```

Once your download and installation are completed, you need to do two things:

1) Create a valid configuration file for Prosody
2) Create users that you want to be in your chat list

Obviously, if you want to expose your chat service through the Internet, you will need to setup port-forwarding on your router and point to the selected port for the Prosody server. We'll come to that in a minute.

You can locate the config file in:

```
/etc/prosody/prosody.cfg.lua
```

Create a backup of this file if it already exists. Then, replace the contents with the following. You can also find an example at:
https://prosody.im/doc/example_config

```
modules_enabled = {
	-- Generally required
	"roster"; -- Allow users to have a roster. Recommended ;)
	"saslauth"; -- Authentication for clients and servers.
	"tls"; -- Add support for secure TLS on c2s/s2s connections
	"dialback"; -- s2s dialback support
	"disco"; -- Service discovery
	-- Not essential, but recommended
	"private"; -- Private XML storage (for room bookmarks, etc.)
	"vcard"; -- Allow users to set vCards
	-- Nice to have
	"legacyauth"; -- Legacy authentication. Only used by some old clients and bots.
	"version"; -- Replies to server version requests
	"uptime"; -- Report how long server has been running
	"time"; -- Let others know the time here on this server
	"ping"; -- Replies to XMPP pings with pongs
	"register"; -- Allow users to register on this server using a client and change passwords
	-- Other specific functionality
	"posix"; -- POSIX functionality, sends server to background, enables syslog, etc.
	--"console"; -- telnet to port 5582 (needs console_enabled = true)
	--"bosh"; -- Enable BOSH clients, aka "Jabber over HTTP"
	--"httpserver"; -- Serve static files from a directory over HTTP
};

-- Disable account creation by default, for security
-- For more information see http://prosody.im/doc/creating_accounts
allow_registration = false;

-- Debian:
-- send the server to the background
daemonize = true;

-- Create a pidfile for nohup launch
pidfile = "/var/run/prosody/prosody.pid";

-- Logs info and higher to /var/log
log = {
	-- Log files (change 'infor' to 'debug' for debug logs):
	info = "/var/log/prosody/prosody.log";
	error = "/var/log/prosody/prosody.err";
	-- Syslog:
	{ levels = { "error" }; to = "syslog"; };
}

-- Your personal domain
VirtualHost "mydomain";
c2s_ports = {5222};
```

Let's run through this file and see what it does. The "`modules_enabled`" section defines which modules Prosody loads on startup. Some are required, others are optional. Feel free to select the modules you like. The "`posix`" module for example is necessary for the UNIX system (so, also for Raspberry Pi), while the "`register`" module is useful only if you want to allow external users to register with your chat service (see below).

The next parameter is "`allow_registration`". This allows external registration. I have set this to `false` because I wanted to create my chat service users manually as I want to create a private and

hidden service. Only an administrative user logged onto the Raspberry Pi and with privileges to run Prosody commands can create and delete users.

The "daemonize" parameter is useful, in combination with "pidfile" if you want to run Prosody without creating a nohup script (i.e.: to keep Prosody running even if you log out of the Raspberry Pi). Furthermore, you can manage the Prosody service with the init.d commands (start, restart, stop, etc.) For example:

```
$ sudo /etc/init.d/prosody restart
```

By adding the "log" directive we can detect any error or eventual intrusion in your chat server. You can specify any path you like, but I suggest you keep to the default /var/log files.

The last, "VirtualHost", enables the administrator to create various hosts each with one or more users. A VirtualHost is effectively a single chatroom or chat service, independent from any other, which can can contain one or more users. For my purpose, I created a single domain named "mydomain".

Under each VirtualHost you can define zero or more directives that can override the master ones: you can specify a specific log directive, or allow external registration by overriding the parameter "allow_registration" for example.

In my file I explicitly put the "c2s_ports" directive with the default value of 5222 in order to allow me to easily change it as we will see in the next section. That directive specifies a comma-separated array of port numbers on which the Prosody server will listen.

Create users

Prosody comes with an utility called prosodyctl that allows you to create users and manage the service. Create your users with the following, entering a password for the user in each case when prompted:

```
$ sudo prosodyctl adduser user1@mydomain
$ sudo prosodyctl adduser user2@mydomain
```

By entering the above, we have created two users bound to the 'mydomain' domain.

Now, restart (or start) the server:

```
$ prosodyctl restart
```

Congratulations! Your chat service should now be up and running.

The client side

Now, to test it. Let's install some XMPP clients on your preferred devices. I have an Android phone, on which I installed Xabber (from Play Store) and a MacBook with Adium.

Both Xabber and Adium have a simple wizard to configure your account. With Xabber, you create a new account with username user1@mydomain and the password you specified.

Other important parameters you need to modify are the **host** and **port** under Settings > XMPP accounts panel. By default, Prosody runs on port 5222 (the standard XMPP port), while for the host, simply type your Raspberry Pi's current IP address on your network.

Do the same things with Adium or with another Android device with Xabber (or with CrossTalk for iPhone). Once your devices are connected to the chat service, add each other by adding a new contact. The procedure is the same for all IM services: user1 asks to add user2 and user2 has to confirm the friendship.

You should be able to send and receive messages from user1 and user2, through your XMPP clients and via your Raspberry Pi.

Expose the service to the Internet

Optionally you can publish your service onto the Internet. First, choose a port (or a set of ports) that you can forward behind your router (default: 5222), noting that some Internet providers restrict access to some ports. For example I set the directive like this:

```
c2s_ports = { 9999 }
```

and port-forwarded 9999 to my local Raspberry Pi's IP address. In this way, all requestes received through port 9999 from the Internet will be directed to my Raspberry Pi, which Prosody will catch as it is listening on port 9999.

To access the chat service across the Internet from outside your LAN change the host and port for Xabber: change the IP address to your router's public

facing IP address and port to 9999. You should now be able to exchange messages outside your local network. Try it!

Security

Once you have exposed your service you are probably asking yourself: how can I protect my chatrooms with a secure channel? Prosody allows you to set up SSL/TLS configuration to encrypt communications and to ensure that the server you are talking with is really your server (the Raspberry Pi).

To add an advanced security configuration you first need to create a certificate, which is a kind of bucked in which is stored the server's identity. With a trusted certificate successfully verified a client can check if the server is really who it claims to be. To create a certificate and a valid SSL/TLS configuration I suggest these links:

https://prosody.im/doc/certificates
https://prosody.im/doc/advanced_ssl_config

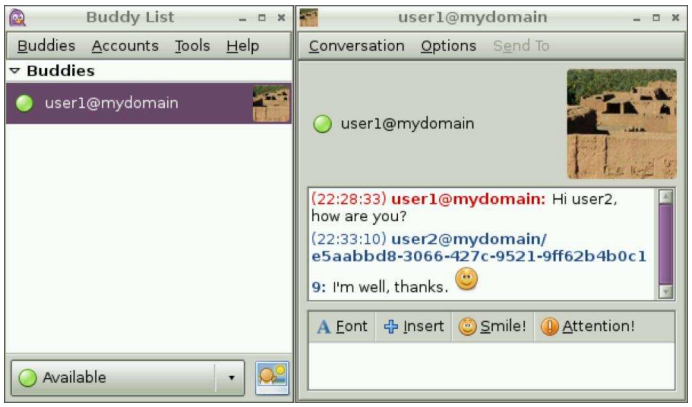
References

Prosody website: https://prosody.im

Xabber client: http://www.xabber.org
Adium client: https://adium.im

The excellent Pidgin chat client is available directly from the Raspbian respository for use on the Raspberry Pi:

```
$ sudo apt-get install pidgin
```



After installation you will find Pidgin under LXDE's Main Menu / Internet / Pidgin Instant Messenger. Within Pidgin, the menu option Accounts / Manage Accounts lets you add one of your pre-defined Prosody users.

UPiS module

The all-in-one solution

A wealth of features in a single board:

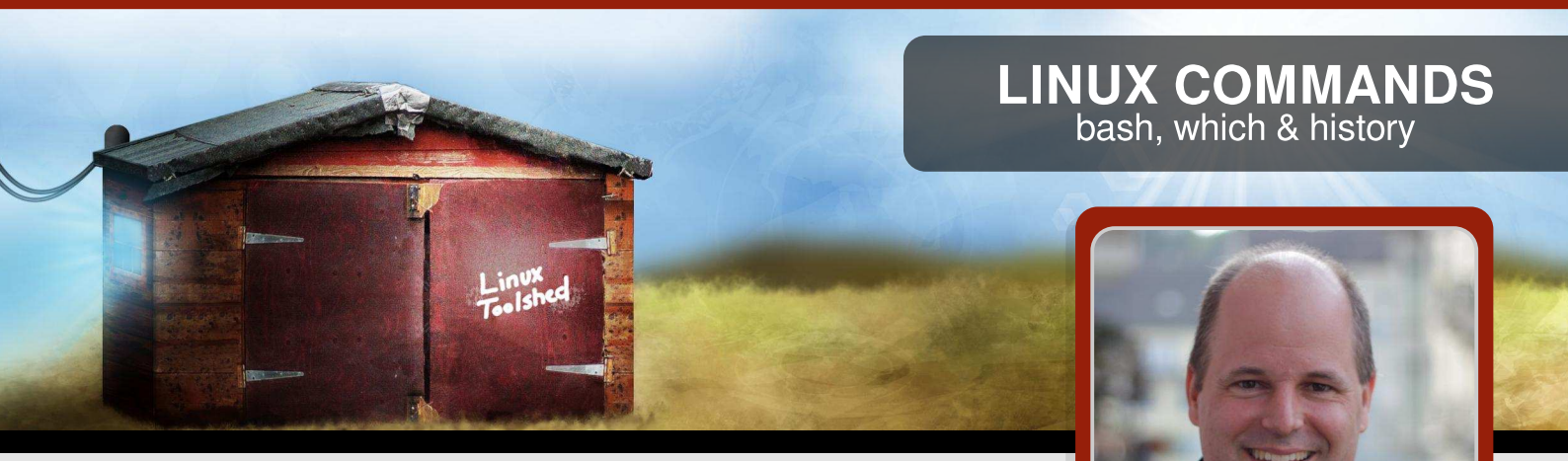
- UPS function with Li-Po battery
- Versatile powering for RPi (7-18VDC)
- Hardware ON/OFF switch for the RPi
- Software-readable V/mA/°C sensors
- Battery backed-up real-time clock
- Timer controlled RPi ON/OFF
- RS232 and USB interfaces
- I²C PiCO interface
- 1-wire and iButton input
- Basic I/O pin and relay outputs
- XTEA-based encryption of user software

available from distributors :

πmodules www.pimodules.com

Intelligent modules for your Raspberry Pi

RASPPISHOP
Pi Supply
MODMYPI
NETTO
PI-SHOP.CH
www.m.nu MATTEKNIK FÖR ALLA



Tales from the Linux tool shed:
Don't bash the shell!

Bernhard Suter

Guest Writer

SKILL LEVEL : BEGINNER

Bash

The first Linux command-line command which a user encounters might typically be bash, the default Linux command-line interpreter itself.

For many newcomers, the Linux command-line interface can feel dauntingly complex and often slightly inconsistent. One of the reasons for this feeling might be that there really isn't such a thing as a single monolithic Linux command-line interface. The interface which we see is the result of a relatively basic command-line interpreter shell in combination with a vast and ever growing collection of commands, which are each stand-alone executable programs installed somewhere on the filesystem.

A shell is just like any of these commands, except that one of its main purposes is to interact with the user and to execute other commands. There are several popular shells available - besides the original **sh**, for example also ksh, csh or tcsh - but on Linux today, bash has largely become the standard.

bash is named somewhat tongue in cheek "Bourne again shell", after the sh or Bourne shell, which was written by Steven Bourne, a British computer scientist working at Bell Labs on the early Unix system.

Like most shells, bash can be used both in an interactive as well as in a batch mode. In batch mode, bash becomes a programming language interpreter in its own right. How to use bash as a scripting language is described in more detail in the "Bash Gaffer Tape" series starting in Issue 10 of The MagPi.

In interactive mode, bash allows the user to edit the current command line buffer using the left and right arrow keys, delete etc. and toggle through the history of recent commands using the up and down arrow keys. It also supports many more cryptic but powerful <Ctrl>-key sequences for line editing and navigation. However the main purpose of an interactive shell session is usually to run some other programs on behalf of the user.

Command execution

When the user presses <Enter>, the shell processes the state of the command line editing buffer, breaks the input string into the appropriate pieces, maybe performs some variable substitutions, and executes some built-in commands or sets up one or more external commands to be executed.

For example, in order to execute,

```
pi@raspberrypi ~ $ ls -l $SHELL
-rwxr-xr-x 1 root root 813992 Jan 10 2013
/bin/bash
```

bash first expands the variable “SHELL” to its definition SHELL=/bin/bash, because $<varname> triggers variable substitution. Then it locates the executable which corresponds to the command ls and executes it with the two arguments -l and /bin/bash.

Which

An executable is located by searching in order through the list of directories in the PATH environment variable, which typically in a default installation of Raspbian is set to:

PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games:/usr/games

We can look at where bash would find a particular command by using the which command:

```
pi@raspberrypi ~ $ which ls
/bin/ls
pi@raspberrypi ~ $ which which
/usr/bin/which
```

Each executable program in the search path in fact appears to be a bash command. Many like **ls**, bash itself, or others discussed in this series, are part of the standard Linux distribution. But each user can also write their own programs and run them from the shell like any standard tool. Some commands like cd or history are directly executed by the shell itself and are called "builtin" commands (enter man builtins for a complete list).

I/O redirection & pipelining

One of the most powerful aspects of bash is input/output redirection. Each program which is run from the shell is launched with three pseudo-files open at startup: standard input, standard output and standard error (output). By default they are connected to the terminal where the shell is running. There are two major ways to change that default association: I/O redirection and pipelining.

Redirection

I/O redirection involves the use of operators to connect one of the three I/O streams to a named file instead of the interactive terminal:

< : redirect input
> : redirect output
2> : redirect error output
>> : redirect output and append instead of overwrite

```
pi@raspberrypi ~ $ grep raspberry < input.txt
> output.txt 2> error.txt
```

In the above example, the grep string matching tool is reading lines of text from the file “input.txt”, instead of the console, and writes those which contain the sub-string “raspberry” into the file “output.txt”, instead of printing them to the console. If there are any errors, they are written to the file “error.txt”.

Pipelining

Pipelining requires the shell to run multiple programs in parallel and use the | pipe operator to directly feed the output from one command into the input of the next command, as if they were connected by a data pipe.

Because of a shared design philosophy, most common Linux commands are very simple and typically only have one purpose... and most of them operate on simple unformatted text. Using the pipe operator, the shell allows them to be combined in ingenious ways to achieve much more complex functionality.

To put it into the words of Douglas McIlroy, the inventor of the Unix pipe concept:

“This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.”

```
pi@raspberrypi ~ $ cat /etc/passwd | cut -d:
-f7 | sort | uniq -c | sort -nr
17 /bin/sh
 5 /bin/false
 2 /bin/bash
 1 /usr/sbin/nologin
 1 /bin/sync
```

In this example, the `cat` command reads a file named `/etc/passwd` and simply writes it to the standard output. This output then becomes the input to the **`cut`** command which extracts the 7th field separated by the `:` character and writes this to its output steam. This is then piped to the `sort` command which alphabetically sorts the lines it reads from its input and returns the new sort order to its output. The `uniq` command takes the sorted output and removes and counts the duplicate lines. It writes each line to the output, prefixed with the number of times it was present in the input. Finally this output is passed to a second `sort` command which sorts its input numerically in reverse order and writes this to its output, which happens to be the console from where the shell is running all this.

The `/etc/passwd` file contains the configuration setup of all the user accounts in the system, with different kinds of information separated by a ":" character. The 7th field in each line is the login shell for this user. There are only 2 accounts which use `bash` - the root and pi user accounts in this case. All the other accounts are system or role accounts for particular services and are never intended to be logged in. Some use commands as their login shell, which are not really shells, like `false` or `nologin`.

Job control

While pipelining is used to build up multiple, simple commands into an interconnected super-command, job control allows the user to run several unrelated commands from the same `bash` shell at the same time. This is particularly useful in a graphical environment like the LXDE window desktop, where a command line window can be used to launch several graphical applications:

```
pi@raspberrypi ~ $ debian-reference &
[1] 16220
pi@raspberrypi ~ $ leafpad &
[2] 16236
pi@raspberrypi ~ $ idle3 &
[3] 16256
```

Adding the & to the end of a command line will execute it in the background, immediately freeing up the shell to receive more input and suspending the standard input of the command unless it has been redirected. Any output from the command may continue to go to the shell though. We can also interrupt a blocking command by pressing <Ctrl>-Z to freeze or suspend it and then release it to continue executing in the background by typing `bg`.

If there are any jobs running in the background, we can see their ID and executed command line using the `jobs` command. We can then bring any of them into the foreground context using the command `fg [job number]` and if necessary, terminate the command by pressing <Ctrl>-C.

```
pi@raspberrypi ~ $ jobs
[1] Running debian-reference &
[2] Running leafpad &
[3] Running idle3 &
pi@raspberrypi ~ $ fg 1
debian-reference
^C
pi@raspberrypi ~ $
```

History lesson

Having started this article on the history of `bash`, we are now coming full circle and look at the `history` command. This very useful command lets us look at all the commands we have previously executed and recall some of them in a variety of ways. The `history` command itself lists all the commands which were executed by the current user, as far back as the history file goes. For example:

```
pi@raspberrypi ~ $ history
1 ifconfig -a
2 sudo raspi-config
3 ls
...
```

```
965 cat /etc/passwd | cut -d: -f7 | sort |
uniq -c | sort -nr
966 less /etc/passwd
967 history
pi@raspberrypi ~ $
```

Using the up arrow key, we can the flip back through the history until we get to the command we want to re-run, or edit the line first before running.

If we are looking for something more specific, we can also filter the output of `history`, e.g. with the `grep` tool:

```
pi@raspberrypi ~ $ history | grep grep
909 grep raspberry < input.txt > output.txt 2>
error.txt
911 cat | grep -v mouse | sort | uniq -c
969 history | grep grep
pi@raspberrypi ~ $
```

In a graphical terminal, we can use the mouse to copy/paste any command from the `history` output to the command line, or otherwise recall a line by its number by prefixing it with an exclaimation mark `!`:

```
pi@raspberrypi ~ $ !909
grep raspberry < input.txt > output.txt 2>
error.txt
-bash: input.txt: No such file or directory
pi@raspberrypi ~ $
```

We can also press <Ctrl>-R to start a search back through the history for the first command which contains whatever we are now typing:

```
pi@raspberrypi ~ $
(reverse-i-search)`fun': echo “bash is a lot
of fun…”
```

Auto-complete

Another convenient shorthand is contextual auto-completion. Depending on the position in the command-line, pressing the <Tab> key once may complete the current word behind the cursor, if there is a single matching completion. Pressing the <Tab> key twice prints all the possible suggestions. For example, typing `ba`<Tab><Tab> results in:

```
pi@raspberrypi ~ $ ba
badblocks base64 basename bash bashbug
pi@raspberrypi ~ $ ba
```

Auto-completion is a nice way to avoid typos and save on some typing for command names or complex filesystem paths for example. For some common commands, even the options can be auto-completed. For example, typing `ls --`<Tab><Tab> results in:

```
pi@raspberrypi ~ $ ls --
--all
--almost-all
...
--version
--width=
pi@raspberrypi ~ $ ls --
```

Simplicity

You can also customize the behaviour and appearance of your login shell session by adding configuration commands to some of the default files which bash loads and executes on startup, e.g. reading from `~/.bashrc`.

Part of the elegant simplicity of the Unix command model is that the command line interpreter is nothing but a command itself, which happens to execute other commands. It has no special privileges and anybody could write one of their own. However, a well designed shell like `bash` should play to the strength of the Unix philosophy where each tool should be simple, do one thing well and be able to act as a filter in a multi-stage pipeline of commands to implement complex functions as needed.

Modern shells like `bash` have a lot of nice usability features which make interacting with a Linux terminal session a lot less daunting and knowing some of the tricks and shorthands can save a lot of time.

But `bash` is only as useful as the commands it can execute. What are some of your favorite or most puzzling Linux commands? Send your comments to editor@themagpi.com and maybe we can cover them in a later episode.

The MagPi What's On Guide

Want to keep up to date with all things Raspberry Pi in your area?

Then this section of The MagPi is for you! We aim to list Raspberry Jam events in your area, providing you with a Raspberry Pi calendar for the month ahead.

Are you in charge of running a Raspberry Pi event? Want to publicise it?
Email us at: editor@themagpi.com

Focus on Education @ Cambridge Raspberry Jam

When: Saturday 10th May 2014, 12.00pm to 6.00pm
Where: Institute of Astronomy, Madingley Rd, CB3 0HA, UK

For teachers, co-ordinators and other educators: talks and presentations centring on the new Computing curriculum. http://www.eventbrite.co.uk/e/10867072707

North Staffordshire Raspberry Jam

When: Monday 12th May 2014, 6.00pm to 9.00pm
Where: Newcastle under Lyme College, Knutton Lane, Newcastle-under-Lyme, ST5 2GB, UK

An evening to allow Raspberry Pi users an opportunity to network, learn and show off projects.
https://northstaffsrjam1.eventbrite.co.uk

PiCymru - Cardiff Raspberry Jam

When: Saturday 17th May 2014, 1.30pm to 5.30pm
Where: Howardian Centre, Hammond Way, Cardiff, CF23 9NB, UK

The second event organised by PiCymru: talks, demos and show and tell. The MagPi will be there - come along and say hello. https://www.eventbrite.co.uk/e/11039175471

Maker Faire Bay Area 2014

When: Saturday 17th to Sunday 18th May 2014, 10.00am to 6.00pm (PDT)
Where: San Mateo County Event Center, 1346 Saratoga Dr, San Mateo, CA 94403, USA

Celebrate MAKE magazine's 17th Maker Faire, showcasing creativity in the areas of science and technology, engineering, food, and arts and crafts. http://www.eventbrite.com/e/9098302267

Using Raspberry Pi GPIO to teach computing

When: Wednesday 21st May 2014, 1.30pm to 5.30pm
Where: East Building 0.10, The University of Bath, Claverton Down, Bath, BA2 7AY, UK

A half day course on getting the most from Raspberry Pis in the school environment.
http://www.eventbrite.co.uk/e/11167882437

Expand your Pi

Stackable Raspberry Pi expansion boards and accessories

ADC-DAC Pi

2x 12 bit analogue to digital channels and 2x 12 bit digital to analogue channels.

IO Pi

32 digital input/output channels for your Raspberry Pi. Stack up to four IO Pi boards to give you 128 I/O channels.

RTC Pi

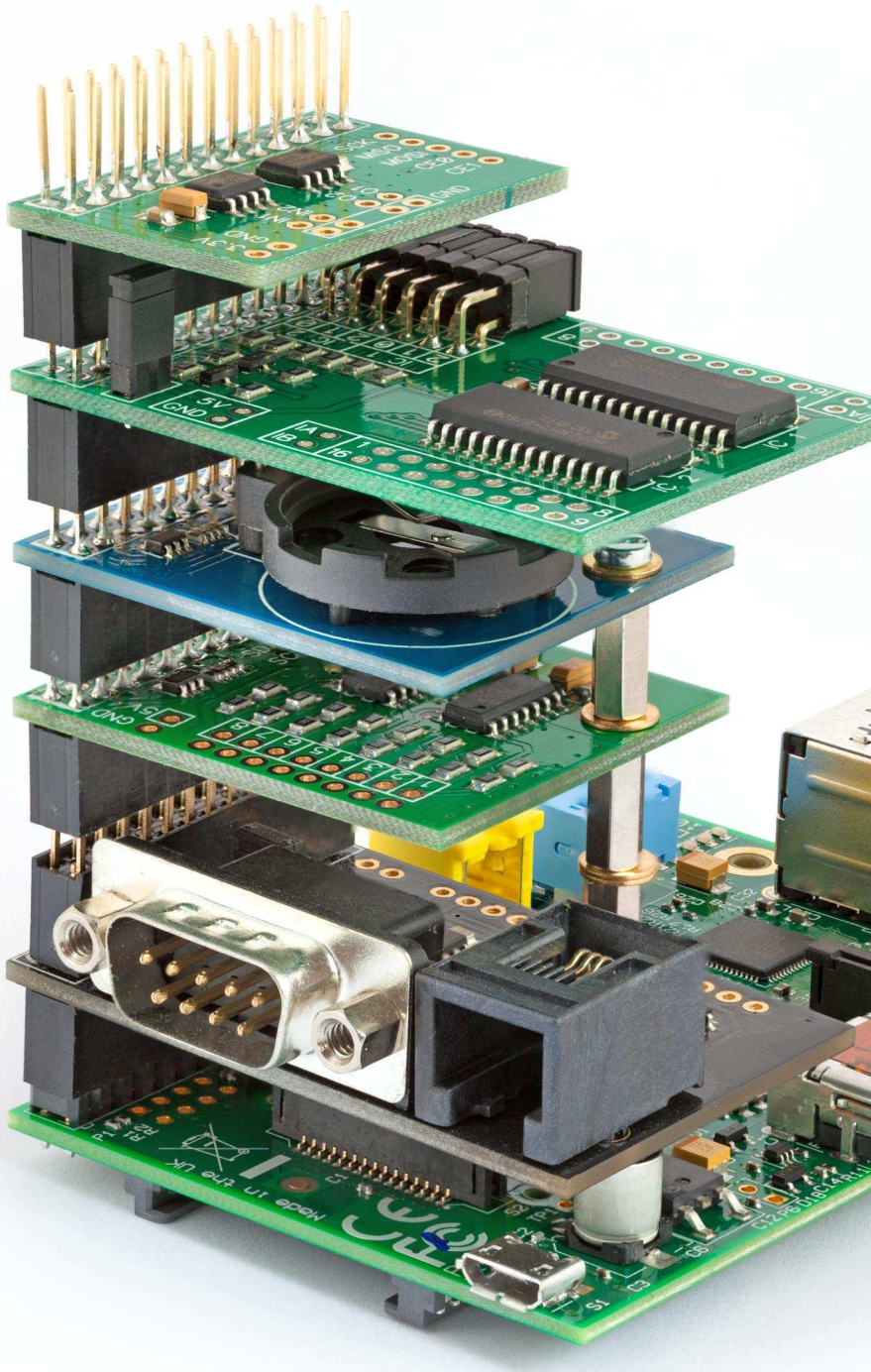
Real-time clock with battery backup and 5V I²C level converter for adding external 5V I²C devices to your Raspberry Pi.

ADC Pi

8 channel analogue to digital converter. I²C address selection allows you to add up to 32 analogue channels to your Raspberry Pi.

Com Pi

RS232 and 1-Wire® expansion board adds a serial port to your Raspberry Pi. Ideal for the Model A to enable headless communication.



AB electronics UK

www.abelectronics.co.uk

SONIC PI 2.0

Get your groove on!

Discover new samples, synths, studio effects and Live Coding

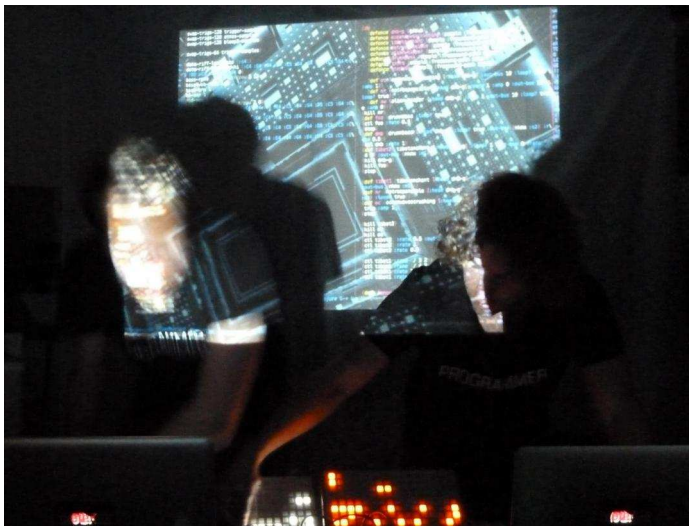
Samuel Aaron

Guest Writer

SKILL LEVEL : BEGINNER

Live Coding

The laser beams sliced through the wafts of smoke as the subwoofer pumped bass deep into the bodies of the crowd. The atmosphere was ripe with a heady mix of synths and dancing. However something wasn't quite right in this nightclub. Projected in bright colours above the DJ booth was futuristic text, moving, dancing, flashing. This wasn't fancy visuals, it was merely a projection of a terminal containing Emacs. The occupants of the DJ booth weren't spinning disks, they were writing, editing and evaluating code. This was a Meta-eX (http://meta-ex.com) gig. The code was their musical interface and they were playing it live.



Coding music like this is a growing trend and is often described as Live Coding (http://toplap.org). One of the recent directions this approach to music making has taken is the Algorave (http://algorave.com) - events where artists code music for people to dance to.

However, you don't need to be in a nightclub to Live Code. As one half of Meta-eX and author of Sonic Pi, I designed version 2 to give you the power to Live Code music anywhere you can take your Raspberry Pi and a pair of headphones, or some speakers. Once you reach the end of this article, you'll be programming your own beats and modifying them live. Where you go afterwards will only be constrained by your imagination.

Getting Sonic Pi v2.0

For this article you will need version 2.0 of Sonic Pi. You can tell if you are using version 2.0 from the opening splash screen. At the time of writing version 2 is in the final stages of development. You can get a copy of the latest release candidate, along with installation instructions, at http://sonic-pi.net/get-v2.0. When version 2.0 is released, you will be able to update with:

```
sudo apt-get install sonic-pi
```

You will then be able to open Sonic Pi by clicking on the main menu, and looking within `Education -> Sonic Pi`.

What's New?

Some of you may have already had a play around with Sonic Pi. Hopefully you all had fun making beeps of different pitches. You can take all the music coding skills you've learned with version 1 and apply it to version 2. For those of you that have yet to open Sonic Pi, now is definitely the time to give it a try. You'll be amazed with what you can do with it. Here's a quick list of the major new features:

* Ships with over 20 synth sounds
* Ability to play any wav or aiff sample file
* Ships with over 70 samples
* Extremely accurate timing
* Support for over 10 studio effects: reverb, echo, distortion, etc.
* Support for Live Coding: changing the code while it runs

Let's have a look at all of these features.

Playing a drum loop

Let's code up a simple drum loop. We can use the *amen* break to get us started. In the main code editor window of Sonic Pi, type the following and then hit the Run button:

```
sample :loop_amen
```

Boom! Instant drums! Go on, press it a few times. Have fun. I'll still be here when you've finished.

But that's not all. We can mess around with the sample. Try this:

```
sample :loop_amen, rate: 0.5
```

Oooh, half speed. Go on, try changing the rate. Try lower and higher numbers. What happens if you use a negative number?

What if we wanted to play the loop a few times over? One way to do this is to call sample a few times with some sleeps in between:

```
sample :loop_amen
sleep 1.753
sample :loop_amen
sleep 1.753
sample :loop_amen
```

However, this could get a bit annoying if you wanted to repeat it 10 times. So we have a nice way of saying that with code:

```
10.times do
  sample :loop_amen
  sleep 1.753
end
```

Of course, we can change the 10 to whatever number we want. Go on, try it! What if we want to loop forever? We simply say `loop` instead of `10.times`. Also, I'm sure you're asking what the magic 1.753 represents and how I got it. Well, it is the length of the sample in seconds and I got it because I asked Sonic Pi:

```
puts sample_duration :loop_amen
```

And it told me 1.753310657596372 - I just shortended it to 1.753 to make it easier for you to type in. Now, the cool thing is, we can combine this code and add a variable for fun:

```
sample_to_loop = :loop_amen
sample_rate = 0.5

loop do
  sample sample_to_loop, rate: sample_rate
  sleep sample_duration sample_to_loop, rate:
sample_rate
end
```

Now, you can change the `:loop_amen` to any of the other loop samples (use the auto-complete to discover them). Change the rate too. Have fun!

For a complete list of available samples click the Help button.

Adding Effects

One of the most exciting features in version 2.0 of Sonic Pi is the support for studio effects such as reverb, echo and distortion. These are really easy to use. For example take the following sample trigger code:

```
sample :guit_e_fifths
```

To add some reverb to this, we simply need to wrap it with a `with_fx` block:

```
with_fx :reverb do
  sample :guit_e_fifths
end
```

To add some distortion, we can add more fx:

```
with_fx :reverb do
  with_fx :disortion do
    sample :guit_e_fifths
  end
end
```

Just like synths and samples, FX also supports parameters, so you can tinker with their settings:

```
with_fx :reverb, mix: 0.8 do
  with_fx :distortion, distort: 0.8 do
    sample :guit_e_fifths
  end
end
```

Of course, you can wrap FX blocks around any code. For example here's how you'd combine the `:ixi_techno` FX and our drum loop:

```
with_fx :ixi_techno do
  loop do
    sample :loop_amen
    sleep sample_duration :loop_amen
  end
end
```

For a complete list of FX and their parameters click the Help button.

Live Coding a Synth Loop

Now we've mastered the basics of triggering samples, sleeping and looping, let's do the same with some synths and then jump head first into live coding territory:

```
loop do
  use_synth :tb303
  play 30, attack: 0, release: 0.3
  sleep 0.5
end
```

So, what do the numbers mean in this example? Well, you could stop it playing, change a number, then start it and see if you can hear the difference. However all that stopping and starting gets quite annoying. Let's make it possible to live code so you can instantly hear changes. We need to put our code into a named function which we loop:

```
define :play_my_synth do
  use_synth :tb303
  play 30, attack: 0, release: 0.3
  sleep 0.5
end

loop do
  play_my_synth
end
```

Now when you run this it will sound exactly the same as the simpler loop above. However, now we have given our code a name (in this case, `play_my_synth`) we can change the definition of our code while things run. Follow these steps:

1. Write the code above (both the define and loop blocks)
2. Press the Run button
3. Comment out the loop block (by adding # at the beginning of each line)
4. Change the definition of your function
5. Press the Run button again
6. Keep changing the definition and pressing Run!
7. Press Stop

For example, start with the code above. Hit Run. Comment out the `loop` block then change the `play` command to something different. Your code should look similar to this:

```
define :play_my_synth do
  use_synth :tb303
  play 45, attack: 0, release: 0.3, cutoff:
70
  sleep 0.5
end

# loop do
#   play_my_synth
# end
```

Then hit the Run button again. You should hear the note go higher. Try changing the attack, release and cutoff parameters. Listen to the effect they have. Notice that attack and release change the length of the note and that cutoff changes the 'brightness' of the sound. Try changing the synth too - fun values are `:prophet`, `:dsaw` and `:supersaw`. Press the Help button for a full list.

There are lots of other things we can do now, but unfortunately I'm running out of space in this article so I'll just throw a couple of ideas at you. First, we can try some randomisation. A really fun function is `rrand`. It will return a random value between two values. We can use this to make the cuffoff bounce around for a really cool effect. Instead of passing a number like 70 to the cutoff value, try `rrand(40, 120)`. Also, instead of only playing note 45, let's choose a value from a list of numbers. Try changing 45 to `chord(:a3, :minor).choose`. Your play line should look like this:

```
play chord(:a2, :minor).choose, attack: 0,
release: 0.3, cutoff: rrand(40, 120)
```

Now you can start experimenting with different chords and range values for cutoff. You can do something similar with the pan value too:

```
play chord(:a2, :minor).choose, attack: 0,
release: 0.3, cutoff: rrand(40, 120), pan:
rrand(-1, 1)
```

Now throw some FX in, mess around and just have fun! Here are some interesting starting points for you to play with. Change the code, mash it up, take it in a new direction and perform for your friends!

```
# Haunted Bells
loop do
  sample :perc_bell, rate: (rrand 0.125,1.5)
  sleep rrand(0.5, 2)
end
```

```
# FM Noise
use_synth :fm
loop do
  p = play chord(:Eb3, :minor).choose -
[0, 12, -12].choose, divisor: 0.01, div_slide:
rrand(1, 100), depth: rrand(0.1, 2), attack:
0.01, release: rrand(0.1, 5), amp: 0.5
  p.control divisor: rrand(0.001, 50)
  sleep [0.5, 1, 2].choose
end
```

```
# Driving Pulse
define :drums do
  sample :drum_heavy_kick, rate: 0.75
  sleep 0.5
  sample :drum_heavy_kick
  sleep 0.5
end
define :synths do
  use_synth :mod_pulse
  use_synth_defaults amp: 1, mod_range: 15,
attack: 0.03, release: 0.6, cutoff: 80,
pulse_width: 0.2, mod_rate: 4
  play 30
  sleep 0.25
  play 38
  sleep 0.25
end
in_thread(name: :t1){loop{drums}}
in_thread(name: :t2){loop{synths}}
```

```
#tron bike
loop do
  with_synth :dsaw do
    with_fx(:slicer,freq: [4,8].choose) do
      with_fx(:reverb,room: 0.5, mix: 0.3) do
        n1 = chord([:b1, :b2, :e1, :e2, :b3,
:e3].choose, :minor).choose
        n2 = chord([:b1, :b2, :e1, :e2, :b3,
:e3].choose, :minor).choose
        p = play n1, amp: 2, release: 8,
note_slide: 4, cutoff: 30, cutoff_slide: 4,
detune: rrand(0, 0.2)
        p.control note: n2, cutoff: rrand(80,
120)
      end
    end
  end
  sleep 8
end
```

Feedback & Question Time

At The MagPi, we love to hear your feedback - whether it is by email, Facebook, Twitter or face to face at one the many Raspberry Pi related events that are happening globally. Recently, our very own Colin Deady was at the Digimakers At-Bristol event and here is some of the feedback he received from people visiting The MagPi stand:

Every month the magazine is released some of the kids at my school print and photocopy it so they each have their own copy.

Keep up the good work, we look forward to every issue.

From a teacher

It's great to see Volume 2 of The MagPi in print. It's looking even better than Volume 1.

Anon

The MagPi magpie... it looks more like an eagle

Anon

Do you cover Minecraft?

Teen 1

Do you cover Minecraft?

Teen 2

Do you cover Minecraft?

Teen ...n

As you can see there was a common thread to a number of these questions - and the answer to those is YES! (Issue 11 covers Minecraft and is available online and in print.)

So the next time you're at an event and you see The MagPi stand, come over and say hello!

I recently discovered the Raspberry Pi while looking for an FM transmitter I could use at work after getting tired of listening to the same songs I've been listening to for the last 35 years. I found an easy solution using the Rasberry Pi. I also discovered your fantastic magazine, which brought back great memories of typing in code from magazines into my Vic 20. I have always regretted selling that computer, and the Raspberry Pi is a great way to bring back those days.

I look forward to getting to know my little computer better through your magazine. I have an exciting project in mind, and can't wait to get experienced enough so I can implement and share it!

Thank you,
Lynn Willis
Salt Lake City, Utah

If you are interested in writing for The MagPi, would like to request an article or would like to join the team involved in the production of the magazine, please get in touch by emailing the editor at:

editor@themagpi.com

The MagPi is a trademark of The MagPi Ltd. Raspberry Pi is a trademark of the Raspberry Pi Foundation. The MagPi magazine is collaboratively produced by an independent group of Raspberry Pi owners, and is not affiliated in any way with the Raspberry Pi Foundation. It is prohibited to commercially produce this magazine without authorization from The MagPi Ltd. Printing for non commercial purposes is agreeable under the Creative Commons license below. The MagPi does not accept ownership or responsibility for the content or opinions expressed in any of the articles included in this issue. All articles are checked and tested before the release deadline is met but some faults may remain. The reader is responsible for all consequences, both to software and hardware, following the implementation of any of the advice or code printed. The MagPi does not claim to own any copyright licenses and all content of the articles are submitted with the responsibility lying with that of the article writer. This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License. To view a copy of this license, visit:

http://creativecommons.org/licenses/by-nc-sa/3.0/

Alternatively, send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.